Vol.9 No.5 October 1990

BEEBUG

FOR THE BBC MICRO & MASTER SERIES

Spinning Text

- INITIALISING ROM IMAGES
- PINPOINTING A DATE
- INTERWORD AND INTERSHEET
- DATABASES

BEEBUG Vol.9 No.5 October 1990

FEATURES

REVIEWS

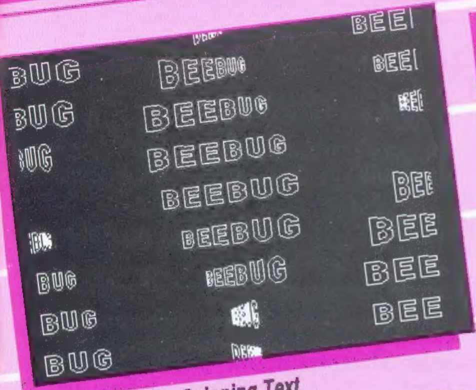
REGULAR ITEMS

HINTS & TIPS

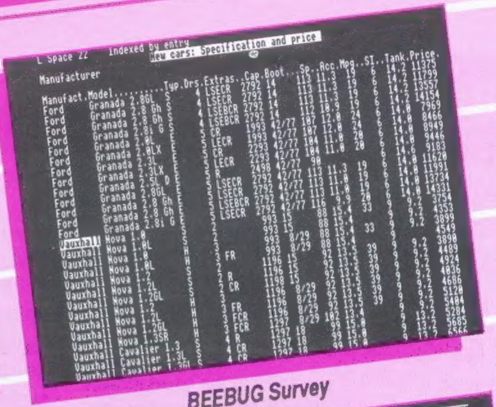
PROGRAM INFORMATION

All listings published in BEEBUG magazine are produced directly from working programs. They are formatted using LISTO 1 and WIDTH 40. The space following the line number is to aid readability only, and may be omitted when the program is typed in. However, the rest of each line should be entered exactly as printed, and checked carefully. When entering a listing, pay special attention to the difference between the digit one and a lower case l (L). Also note that the vertical bar character (Shift \) is reproduced in listings as |.

All programs in BEEBUG magazine will run on any BBC micro with Basic II or later, unless otherwise indicated. Members with Basic I are referred to the article on page 44 of BEEBUG Vol.7 No.2 (reprints

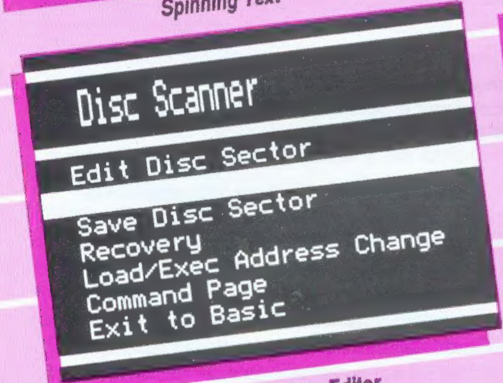


Spinning Text



```
L Space 22    Indexed by entry   New cars: Specification and price

Manufacturer

Manufact.Model..........Typ.Drs.Extras..Cap.Boot...Sp..Acc.Mpg..SI..Tank.Price.
Ford      Granada 2.8GL  S   4  LSECR   2792 14     113 11.3 19   6  14.2 11375
Ford      Granada 2.8 Gh S   4  LSECR   2792 14     113 11.3 19   6  14.2 11799
Ford      Granada 2.8 Gh S   4  LSEBCR  2792 14     113 11.3 19   6  14.2 13557
Ford      Granada 2.8i G S   4  LSEBCR  2792 14     112 10.9 19   6  14.2 14154
Ford      Granada 2.0L   E   5  CR      1993 42/77  107 12.0 24   6  14.0  7969
Ford      Granada 2.0LX  E   5  LECR    1993 42/77  107 12.0 24   6  14.0  8466
Ford      Granada 2.3L   E   5  CR      2293 42/77  104 11.0 20   6  14.0  8949
Ford      Granada 2.3LX  E   5  LECR    2293 42/77  104 11.0 20   6  14.0  9446
Ford      Granada 2.5L D E   5  R       2498 42/77   98           6  14.6  9183
Ford      Granada 2.8GL  E   5  LSECR   2792 42/77  113 11.3 19   6  14.0 11620
Ford      Granada 2.8 Gh E   5  LSECR   2792 42/77  113 11.0 19   6  14.0 11991
Ford      Granada 2.8 Gh E   5  LSEBCR  2792 42/77  113 11.0 19   6  14.0 13734
Ford      Granada 2.8i G E   5  LSECR   2792 42/77  116  9.9 20   6  14.0 14331
Vauxhall  Nova 1.0       S   2           993 15      88 15.4 33   9   9.2  3754
Vauxhall  Nova 1.0L      S   2           993 15      88 15.4      9   9.2  4353
Vauxhall  Nova 1.0       H   3           993 8/29    88 15.4 33   9   9.2  3899
Vauxhall  Nova 1.0L      H   3  FR       993 8/29    88 15.4      9        4549
Vauxhall  Nova 1.2       S   2          1196 15      92 13.5 39   9   9.2  3890
Vauxhall  Nova 1.2L      S   2  R       1196 15      92 13.5 39   9   9.2  4489
Vauxhall  Nova 1.2GL     S   2  CR      1198 15      92 13.5 39   9   9.2  4924
Vauxhall  Nova 1.2       H   3          1196 8/29    92 13.5 39   9   9.2  4036
Vauxhall  Nova 1.2L      H   3  FR      1196 8/29    92 13.5 39   9   9.2  4686
Vauxhall  Nova 1.2GL     H   3  FCR     1196 8/29    92 13.5 39   9   9.2  5120
Vauxhall  Nova 1.3SR     H   3  FCR     1297 8/29   102 13.4      9   9.2  5404
Vauxhall  Cavalier 1.3   S   4  R       1297 18      99 15.0      9  13.2  5284
Vauxhall  Cavalier 1.3L  S   4  CR      1297 18      99 15.0      9  13.2  5685
Vauxhall  Cavalier 1.3GL S   4  CR      1297 18      99 15.0      9  13.2  6566
```

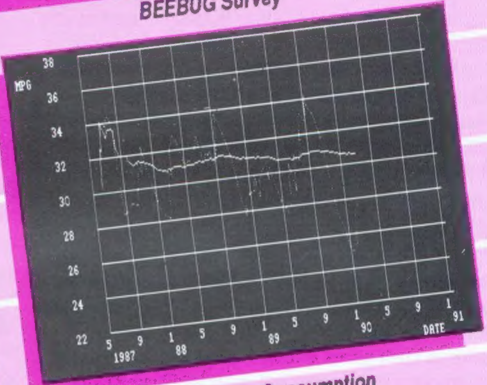
BEEBUG Survey



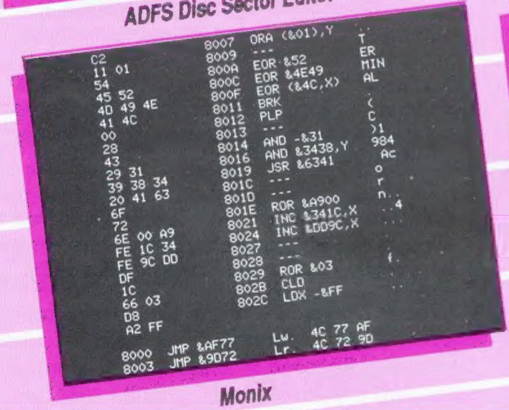
Disc Scanner

Edit Disc Sector
Save Disc Sector
Recovery
Load/Exec Address Change
Command Page
Exit to Basic

ADFS Disc Sector Editor

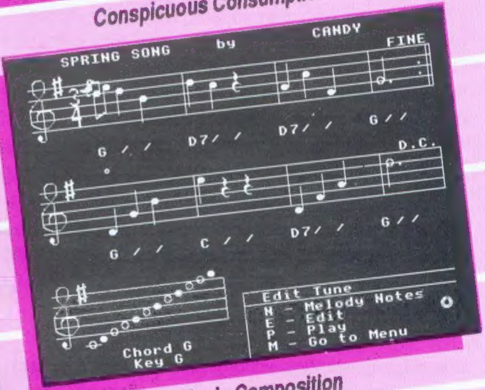


Conspicuous Consumption



```
                 8007  ORA (&01),Y     T
C2               8009  ---             ER
11 01            800A  EOR &52         MIN
54               800C  EOR &4E49       AL
45 52            800F  EOR (&4C,X)     .
4D 49 4E         8011  BRK             (
41 4C            8012  PLP             C
00               8013  ---             )1
28               8014  AND -&31        984
43               8016  AND &3438,Y      Ac
29 31            8019  JSR &6341       o
39 38 34         801C  ---             r
20 41 63         801D  ---             n..
6F               801E  ROR &A900       ..4
72               8021  INC &341C,X     ...
6E 00 A9         8024  INC &DD9C,X
FE 1C 34         8027  ---
FE 9C DD         8028  ---             f.
DF               8029  ROR &03
1C               802B  CLD             ..
66 03            802C  LDX -&FF
D8
A2 FF
8000  JMP &AF77        Lw.  4C 77 AF
8003  JMP &9D72        Lr.  4C 72 9D
```

Monix



Edit Tune
Melody Notes
N - Edit
E - Play
P - Go to Menu
M

Music Composition

available on receipt of an A5 SAE), and are strongly advised to upgrade to Basic II. Any second processor fitted to the computer should be turned off before the programs are run.

Where a program requires a certain configuration, this is indicated by symbols at the beginning of the article (as shown opposite). Any other requirements are referred to explicitly in the text of the article.

Program will not function on a cassette-based system.

Program needs at least one bank of sideways RAM.

Program is for Master 128 and Compact only.

Editor's Jottings

RETAIL CATALOGUE

With this issue of BEEBUG you will find the latest version of our Retail Catalogue containing details of Acorn related products. We are producing two additional catalogues, one on Commodore Amiga products, and one on Business products including Amstrad PCs PCWs and Fax machines. Please let us know if you would like to be put on the mailing list to receive either of these. Associated with the new catalogue are some fundamental changes to our policy which we believe will improve the service we offer.

We recognise that in the past our members' prices have sometimes been higher than the discounted prices offered by some other dealers. Our ability to match or even improve on such prices is linked to our purchasing power. From now on a single price will apply to all customers, which we hope, with your support, will enable us to set even better prices in the future. Consequently our Retail Catalogue will just have one price for all products from now on. However, for your convenience we will be showing the price both exclusive and inclusive of VAT.

Members will continue to benefit from the discounts that we have always offered on our own software products. This applies to magazine products like Edikit and Astaad, and to software products like BEEBUG C and Masterfile. Members' prices on these products will be listed each month in the magazine.

It is also our intention where possible to offer members special offers on selected items. These offers will normally be valid only for one month, i.e. until the next issue of BEEBUG arrives. Members' prices and any special offers will feature regularly in the magazine.

TECHNICAL SUPPORT

We have also taken steps to improve the quality of help and advice available from our enlarged and restructured Technical Support team. Jeffrey George is the new manager of this group, and technical advice will be available from him and team members Graeme Davidson, Gary Blackwell, Nathan Brown, and Darren Finch. Secretarial and administrative back-up is provided by Barbara Oliver.

Robert Barnes, who was previously Technical Manager is now Sales Manager, and team member Ian MacDougall has moved to Software Development managed by John Wallace.

The new Technical Support team will endeavour to deal with all your enquiries more quickly and more efficiently than ever before. We will in future, concentrate on providing in-depth technical advice to members who have queries with products they have purchased from us. We will also endeavour to provide general help and advice so that you may continue to get the best use from your system. Regrettably, we cannot continue to provide support on products purchased from other dealers. Please bear in mind, that unlike other magazines and organisations, we make no extra charge for the help and support which our Technical Support team provides. Our Returns and Repairs departments have also been supplemented by extra staff to improve service here.

All the changes described above have been introduced to ensure that we can continue to provide the best service to all users of the BBC micro and Master series, and to BEEBUG members in particular.

BEEBUG OPEN DAY

We shall be holding Open Days this year on Sunday 7th October and Sunday 25th November, from 10am to 4pm. This provides the ideal opportunity to see and try out a wide variety of hardware and software, and there should be a few bargains to be had as well. Further details are included with this issue of BEEBUG.

News News News News News News

NEW FLAGSHIP ARCHIMEDES

At the Acorn User Show Acorn announced a new flagship computer in the Archimedes range. Designated the A540 the new machine features a completely redesigned main board with separate plug in processor board containing the latest ARM3 processor chip, 4MBytes of cache memory, and new FPA (Floating Point Accelerator) chip for enhanced performance of 13.5 MIPS (millions of instructions per second). The A540 also features high resolution VGA and SVGA graphics support. With 4MBytes of main memory (expandable to 16MBytes) and 100MByte SCSI hard disc running RISC OS, the new machine costs £2995 plus VAT.

Acorn has also announced an extension to the successful Learning Curve package in the form of the *Archimedes Learning Curve*. This is based on the 2MByte A420 system, complete with all the additional components of The Learning Curve package (1st Word Plus, Genesis, etc.) plus Acorn DTP. The *Archimedes Learning Curve* sells for £1299 plus VAT, and will be supported by a major sales campaign throughout the autumn (dealers will be offering a 'test flight' programme, and a competition with a flight on Concorde as first prize). At the same time, the A420 system has also been reduced in price to £1299 plus VAT, and the A440 system now sells for £1699 plus VAT.

The *Archimedes Learning Curve* is available now, and the A540 from the end of September. Contact you dealer for details. Acorn are at Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, tel. (0223) 245200.

HELP FOR MUSIC 5000 LOVERS

JB Software has published a tutorial book for users of Hybrid Technology's Music 5000 system for the BBC micro. The writer is John Bartlett, a long-time Music 5000 composer, and author of three highly acclaimed Music 5000 albums. The book covers the whole spectrum of users from initial sections for beginners through to advanced topics such as musical arrangement and orchestration. Called *PLAY - the performance tutor for the Music 5000*, the book costs £15.95 inclusive (or £11.95 for self-printing text files on disc) from JB Software, 20 Crawley Avenue, Wellingborough, Northants NN8 3YH, tel. (0933) 675392.

Hybrid Technology has compiled a comprehensive survey of nearly 60 music software packages for the BBC micro. The 20 page report is available for £4.30 inc. p&p from Hybrid Technology Ltd, 273 The Science Park, Cambridge CB4 4WE, tel. (0223) 420360.

MORE SOFTWARE FOR 512 USERS

Essential Software has announced details of its latest product for the 512 co-processor, the *Co-Pro-Filing-System* (CPFS). CPFS is a filing system which sits alongside the DFS or ADFS and turns the 512 into a RAM disc which can then be used by the BBC micro running in native mode. Furthermore, CPFS can take full advantage of both standard and expanded 512 memory. All standard filing system star commands are implemented as well as machine code filing system calls for full compatibility. CPFS will be supplied on EPROM and is expected to cost £24.95 complete from Essential Software, P.O.Box 5, Groby, Leicester LE6 0ZB.

STAR GAZING

Software house Topologika has released details of a new set of interactive programs called *ASTRO* for the exploration of space via computer. There are seven graphics programs and a planet database, and several of the programs focus on specific attainment targets of the Science National Curriculum. ASTRO is available in different versions for the BBC micro, Master 128 and Compact, and the Archimedes range, all at £19.95 inc. VAT. For more details and orders contact Topologika, P.O.Box 39, Stilton, Peterborough PE7 3RL, tel. (0733) 244682.

NEW RESOURCES FOR PRIMARY SCHOOLS

Doncaster based Resource has announced *CASS: Curriculum Analysis Support System* initially for the Primary Curriculum and available on double-sided 80 track disc for the BBC Master system. A classroom pack costs just £29.95. Resource has also given details of a new product for Primary Technology called *Honeypot* (price to be announced) due for release in October, this time for the A3000 and Archimedes range of computers. Further details on both products are available from Resource, Exeter Road, Wheatley, Doncaster, South Yorkshire DN2 4PY, tel. (0302) 340331.

Spinning Text

by Robert Alcock

INTRODUCTION

Spinning Text is a graphics demonstration program. It draws, and smoothly rotates in 3D, any word or short message that you enter.

Animation on the Beeb can often be frustratingly slow especially when large areas are involved. Problems can, however, often be overcome by using various crafty techniques. Spinning Text shows how effective one particular method of animation can be.

ENTERING THE PROGRAM

Type the program in carefully, ensuring that all the line numbers remain unchanged. Special attention should also be paid to the procedures PROCass, PROCdecode and FNcompact. Save the program as 'SpinTxt'. Before running it, set PAGE=&1200 (if using a model B). Now load and run the program.

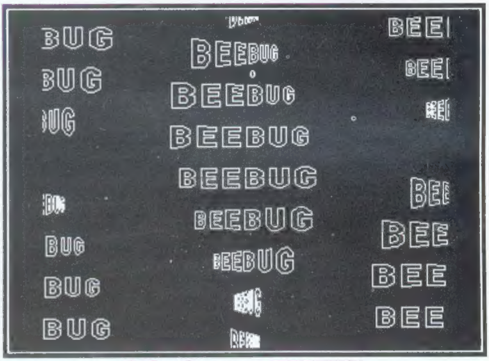
USING THE PROGRAM

When the program is run you are faced with a menu offering four options.

Option 1 (Create Screen) allows you to design your own screen. You are prompted to enter a message of up to seven characters. Any upper case letter or '-','!' or space will be accepted. On pressing return, mode 1 is selected and t[illegible] screen is drawn. This may take up to three minutes depending upon the text entered. Once completed the screen is compacted, so the display will become corrupted for a while. The compacted screen is then saved in directory 'S' using as a file name the word you typed in. For example, if your message is 'HELLO', the screen is saved as 'S.HELLO'. The text is quickly reformed and should begin to spin. Note: ADFS users should create a sub-directory 'S' before running the program.

Selecting option 2 (Display Screen) loads in a screen from disc (or wherever) and in[illegible]ediately starts to animate it. There is no need to specify the directory when asked for the filename, as directory 'S' is assumed.

Once the text is spinning you have control over its direction. The keys 'F' and 'S' make the text spin faster or slower, and pressing Shift as well makes the change more quickly. Space makes the text move either up or down, and Return inverts the direction of rotation of the text. Pressing Escape returns you to the menu.



Spinning text 'BEEBUG'

Option three (Set Colours) allows you to change the colours of any screen that you draw. However, it does not allow you to alter a screen that has already been saved to disc.

To leave the program select option four.

You may also execute any star commands from the menu, but it is not a good idea to try anything which is likely to corrupt memory e.g. *COPY or *BACKUP etc.

At any time if you press Escape you will be returned to the menu. Pressing Shift and Escape will immediately leave the program.

TECHNICAL NOTES

You may notice, when the program is running, that the spinning text is drawn in perspective. This gives a real feeling of depth as the letters move towards and away from the observer. The method used to achieve this is surprisingly easy when you consider each step separately.

The text is created as a 3D model in the yz plane, with its thickness in the x direction (see fig 1). Each corner is rotated about point C by multiplying its co-ordinates by the rotation matrix, i.e.:

```
    ( x') = ( cosA -sinA )( x )
    ( z') = ( sinA  cosA )( z )
```

where x',y',z' are the new and x,y,z are the old co-ordinates with respect to point C. The y co-ordinates are unchanged by this rotation.

Once rotated, the 3D co-ordinates must be transformed so that they lie on the 2D plane representing the monitor screen. If the object lies at point x',y',z' in 3D space its position on the screen will be where the line from that point to the eye intersects the screen (in the xy plane).

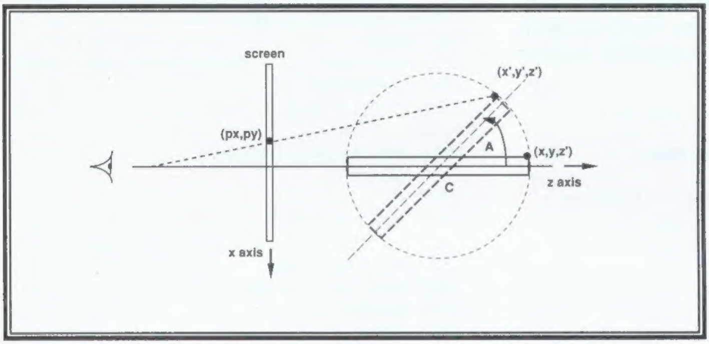


Figure 1. The 3D text model viewed from above

Using the fact that OAA' and OBB' (see fig. 2) are similar triangles, the height AA' can be expressed as:

```
    AA' = BB'* OA/OB.
```

Extending this technique to both x and y co-ordinates gives:

```
    px=x'*OS/OC and py=y'*OS/OC
```

where px and py are the transformed points on the screen (see fig 1). Using these new points the entire image can be drawn in perspective.

Sixteen separate frames in a sequence are then created. In addition, a method of animation must be incorporated that is fast enough to give the appearance of movement. PROCspin and PROCscreen fulfil this function. PROCspin detects the key presses and accordingly alters the step size between each screen scroll to give the impression of different movements of the text. PROCscreen calls the small machine code routine which writes to the screen start register in the 6845 video controller chip.

Earlier I mentioned a compaction routine (i.e. FNcompact). This takes any 20K screen and compresses it, thus taking up less space on your disc. The algorithm is not desperately efficient although it is fast enough to implement through Basic. FNcompact is able to reduce a 20K screen down to between 6K and 10K in about 15 seconds. The function returns the length of the compacted data. PROCdecode takes about 10 seconds to reconstruct the display.

If you wish to use these compaction procedures in your own programs you should add one extra line to each to ensure that there are no variable clashes.

```
 2175 LOCAL A%,B%,D%,N%,S%
 2265 LOCAL A%,D%,S%
```

Also set HIMEM=&2F00 at the start of the program, after selecting the 20K mode you wish to use. This reserves work space for the compaction routine.

You can now simply call FNcompact before you save your screen and call PROCdecode after loading the screen.

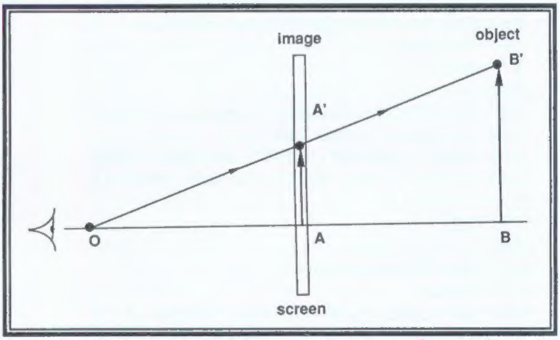


Figure 2. A simple perspective transformation

CHARACTER DATA

Each letter allowed by the program was created on a 9 by 16 grid. To define your own additional characters simply draw out your design on a suitable grid and enter the co-ordinates of the outline. A '-1' before the co-ordinates chosen causes a 'move' to that position . A '-2' signifies the end of the character. Enter your data in the same format as that already used i.e.:

```
    REM 'character'
    DATA xcoord1,ycoord1,xcoord2, , etc.
```

with a gap of 10 between each line number. However, ensure that the very last DATA statement does remain at the end of all the character data. As well as entering the data you must also add your character to the end of the lists of characters in the INSTR command in lines 1050 and 1160. The program should not be renumbered in any way because of the use of a computed RESTORE instruction.

Following these instructions should allow you to extend to a full character set if required. Memory limitations will unfortunately make any extensions for model B users very small, but with PAGE=&E00 on a Master a considerable number of additions can be made.

```
   10 REM Program Spinning Text
   20 REM Version B1.2
   30 REM Author  Robert David Alcock
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
   60 :
  100 HIMEM=&2F00
  110 ON ERROR GOTO190
  120 DIM code 30
  130 DIM C%(3),L%(30,1),ST%(3)
  140 FOR A%=0TO3:READC%(A%):NEXT
  150 FOR A%=0TO3:READST%(A%):NEXT
  160 PROCmenu
  170 END
  180 :
  190 IF ERR=17 ANDINKEY-1 MODE7:GOTO230
  200 IF ERR=17 GOTO 160
  210 MODE 7:REPORT:PRINT" at line ";ERL
  220 PRINT'"Press SPACE to continue":IF
 GET=32 GOTO160
  230 END
  240 :
 1000 DEF PROCcreate
 1010 REPEAT
 1020 INPUTTAB(2,14);SPC(30);TAB(2,14)"E
nter Message : -------";TAB(18,14)text$
 1030 bad$="":found%=TRUE:L%=LEN(text$)
 1040 FOR A%=1 TO L%
 1050 IF INSTR("ABCDEFGHIJKLMNOPQRSTUVWX
YZ-! ",MID$(text$,A%,1))=0 found%=FALSE:
bad$=bad$+MID$(text$,A%,1)
 1060 NEXT
 1070 IF found%=FALSE PRINTbad$;" is not
 found";:L%=0:REPEAT:UNTIL GET:VDU13:PRI
NTSPC(20)
 1080 UNTIL L%<8 AND L%>0
 1090 VDU22,1
 1100 FOR A%=0 TO 3:VDU19,A%,C%(A%);0;:N
EXT
 1110 first%=1:last%=L%:S%=1:frame%=0
 1120 X%=320:Y%=-4
 1130 FOR A=0 TO PI STEP PI/16
 1140 r1=COSA:r2=-SINA:r3=SINA:r4=COSA
 1150 FOR B%=first% TO last% STEP S%
 1160 line%=3000+20*INSTR("ABCDEFGHIJKLM
NOPQRSTUVWXYZ-! ",MID$(text$,B%,1))
 1170 VDU29,X%;Y%;
 1180 PROCedge
 1190 PROCface
 1200 NEXT
 1210 frame%=frame%+1
 1220 IF frame% MOD 8=0 S%=-S%:temp%=fir
st%:first%=last%:last%=temp%
 1230 X%=(X%+640) MOD 1280:Y%=(Y%+576-16
+32*(frame%AND1)) MOD 1024
 1240 NEXT
 1250 FOR A%=0 TO 3:A%?&3000=C%(A%):NEXT
 1260 L%=FNcompact
 1270 OSCLI("SAVE S."+text$+" 2F00 "+STR
$~(L%))
 1280 PROCdecode:PROCspin
 1290 ENDPROC
 1300 :
 1310 DEF PROCedge
 1320 GCOL0,1
 1330 RESTOREline%
 1340 F%=0:Z%=100+L%*50-B%*100
 1350 REPEAT
 1360 READz,y
 1370 IF z=-1 PROCmove:GOTO1360 ELSE IF
z=-2 L%(F%,0)=9999:GOTO1420
 1380 PROCtransform
 1390 PLOT85,px,y:PLOT85,px1,y1
 1400 L%(F%,0)=px1:L%(F%,1)=y1
 1410 F%=F%+1
 1420 UNTIL z=-2
 1430 ENDPROC
 1440 :
 1450 DEF PROCmove
 1460 z=y:READy
 1470 PROCtransform
 1480 MOVEpx,y:MOVEpx1,y1
 1490 L%(F%,0)=9999
 1500 L%(F%+1,0)=px1:L%(F%+1,1)=y1
 1510 F%=F%+2
 1520 ENDPROC
 1530 :
 1540 DEF PROCface
 1550 IF frame% MOD 16=0 ENDPROC
 1560 GCOL0,2
 1570 REPEAT
 1580 IF L%(F%,0)=9999 F%=F%-1:K%=4 ELSE
 K%=5
 1590 PLOTK%,L%(F%,0),L%(F%,1)
 1600 F%=F%-1
 1610 UNTIL F%<=0
 1620 ENDPROC
 1630 :
 1640 DEF PROCtransform
 1650 z=Z%-z*8:y=y*5-40
 1660 px=r1*15+r2*z:px1=r1*-15+r2*z
 1670 pz=r3*15+r4*z:pz1=r3*-15+r4*z
 1680 scale=600/(pz+800):scale1=600/(pz1
+800)
 1690 px=scale*px:px1=scale1*px1
 1700 y1=scale1*y:y=scale1*y
 1710 ENDPROC
 1720 :
 1730 DEF PROCdisplay
 1740 REPEAT
 1750 INPUTTAB(2,14);SPC(30);TAB(2,14);"
Enter Filename - "text$
 1760 UNTILLEN(text$)<8 AND LEN(text$)>0
 1770 VDU22,1
 1780 FOR A%=0 TO 3:VDU19,A%,0;0;:NEXT
 1790 OSCLI("LOAD S."+text$+" 2F00")
 1800 PROCdecode:PROCspin
 1810 ENDPROC
 1820 :
 1830 DEF PROCspin
 1840 FOR A%=0 TO 3:VDU19,A%,A%?&3000;0;
:NEXT
 1850 !&3000=0
 1860 PROCass
 1870 S%=&30A0:B%=0:D%=400
 1880 REPEAT
 1890 PROCscreen(ST%(B%))
 1900 FOR A%=0 TO D%:NEXT
 1910 IF INKEY-99 B%=B% EOR 1
 1920 IF INKEY-74 B%=B% EOR 2
 1930 D%=ABS(D%+((INKEY-1*-4)+1)*10*(INK
EY-68-INKEY-82))
 1940 UNTIL0
 1950 ENDPROC
 1960 :
 1970 DEF PROCscreen(O%)
 1980 S%=S%+O%
 1990 IF S%>=&8000 S%=S%-&5000
 2000 IF S%<&3000 S%=S%+&5000
 2010 ?&70=S% MOD 2048 DIV 8
 2020 ?&71=S% DIV 2048
 2030 CALL code
 2040 ENDPROC
 2050 :
 2060 DEF PROCass
 2070 FOR I%=0 TO 2 STEP 2
 2080 P%=code
 2090 [OPTI%
 2100 LDA#&13:JSR&FFF4
 2110 LDA#&D:STA&FE00:LDA&70:STA&FE01
 2120 LDA#&C:STA&FE00:LDA&71:STA&FE01
 2130 RTS
 2140 ]NEXT
 2150 ENDPROC
 2160 :
 2170 DEF FNcompact
 2180 A%=&2F02:N%=0
 2190 B%=!&3000:!A%=B%:A%=A%+4
 2200 FOR S%=&3004 TO &7FFC STEP4:D%=!S%
 2210 IF D%=B% AND N%<255 N%=N%+1 ELSE ?
A%=N%:N%=0:!(A%+1)=D%:A%=A%+5:B%=D%
 2220 NEXT:?A%=N%
 2230 ?&2F00=A%-4:?&2F01=(A%-4) DIV256
 2240 =A%+1
 2250 :
 2260 DEF PROCdecode
 2270 S%=&7FFC
 2280 FOR A%=(!&2F00 AND &FFFF) TO &2F02
 STEP-5:D%=!A%
 2290 FOR S%=S% TO S%-4*?(A%+4) STEP-4:!
S%=D%:NEXT:NEXT
 2300 ENDPROC
 2310 :
 2320 DEF PROCmenu
 2330 REPEAT
 2340 VDU22,7
 2350 PRINTTAB(0,0);CHR$145;STRING$(36,C
HR$243)
 2360 FOR A%=0 TO 1:PRINTTAB(10,2+A%);CH
R$141;CHR$(131);"Spinning Text":NEXT
 2370 PRINTTAB(0,5);CHR$145;STRING$(36,C
HR$243)
 2380 VDU28,0,24,39,6
 2390 PRINTTAB(12,4);"1 ";CHR$134;"Creat
e screen";TAB(12,6);"2 ";CHR$134;"Displa
y screen";TAB(12,8);"3 ";CHR$134;"Set co
lours";TAB(12,10);"4 ";CHR$134;"Quit"
 2400 PRINTTAB(2,14);"Enter choice"
 2410 REPEAT:K%=GET:UNTIL K%>48 AND K%<5
3 OR K%=42
 2420 IF K%=42:VDU42:INPUT""command$:OSC
LI(command$):PRINT'"Press SPACE to conti
nue":REPEAT:UNTIL GET:GOTO2450
 2430 PRINTTAB(0,(K%-49)*2+4)CHR$136
 2440 IF K%=49 PROCcreate ELSE IFK%=50 P
ROCdisplay: ELSE IFK%=51 PROCsetcolour
 2450 UNTIL K%=52
 2460 ENDPROC
 2470 :
 2480 DEF PROCsetcolour
 2490 CLS
 2500 PRINTTAB(2,2);"Select colour using
 Z and X ,then"'"press return to edit ne
xt colour"
 2510 PRINTTAB(17,6);"Default   Actual"'
TAB(17);"Colour    Colour"
 2520 PRINTTAB(2,9);"Background ";TAB(14
);FNcolour(0)'TAB(2);"Edge Colour";TAB(1
6);FNcolour(1)'TAB(2)"Face Colour";TAB(1
6);FNcolour(7)
 2530 FOR A%=0TO2:PRINTTAB(27,9+A%);FNco
lour(C%(A%)):NEXT
 2540 REPEAT
 2550 FOR A%=0 TO 2
 2560 PRINTTAB(0,9+A%)CHR$136
 2570 REPEAT
 2580 K%=GET
 2590 IF K%=90 C%(A%)=(C%(A%)+1) MOD 8
 2600 IF K%=88 C%(A%)=(C%(A%)+7) MOD 8
 2610 PRINTTAB(27,9+A%);FNcolour(C%(A%))
 2620 UNTIL K%=13
 2630 PRINTTAB(0,9+A%)" "
 2640 NEXT
 2650 PRINTTAB(2,14);"Correct (Y/N)":K%=
GET
 2660 PRINTTAB(2,14);SPC(20)
 2670 UNTIL K%=89
 2680 ENDPROC
 2690 :
 2700 DEF FNcolour(colour)
 2710 RESTORE2790
 2720 FOR C%=0 TO colour:READcol$:NEXT
 2730 col$=col$+STRING$(7-LEN(col$)," ")
 2740 IF colour=0 col$=CHR$135+col$ ELSE
 col$=CHR$(128+colour)+col$
 2750 =col$
 2760 :
 2770 DATA 0,1,7,7
 2780 DATA&2E40,&2BC0,-&2BC0,-&2E40
 2790 DATABlack,Red,Green,Yellow,Blue,Ma
genta,Cyan,White
 3000 :
 3010 REMA
 3020 DATA-1,0,0,0,11,3,16,6,16,9,11,9,0
,6,0,6,5,3,5,3,0,0,0,-1,3,8,3,10,4.5,13,
6,10,6,8,3,8,-2
 3030 REMB
 3040  DATA-1,0,0,0,16,6,16,9,13,9,10,7,
8.5,9,7,9,3,6,0,0,0,-1,3,3,3,7,5,7,6,6,6
,4,5,3,3,3,-1,3,10,3,13,5,13,6,12,6,11,5
,10,3,10,-2
 3050 REMC
 3060 DATA-1,0,3,0,13,3,16,6,16,9,13,9,1
1,6,11,6,12,5,13,4,13,3,12,3,4,4,3,5,3,6
,4,6,5,9,5,9,3,6,0,3,0,0,3,-2
 3070 REMD
 3080 DATA-1,0,0,0,16,6,16,9,13,9,3,6,0,
0,0,-1,3,3,3,13,5,13,6,12,6,4,5,3,3,3,-2
 3090 REME
 3100 DATA-1,0,0,9,0,9,3,3,3,3,7,8,7,8,1
0,3,10,3,13,9,13,9,16,0,16,0,0,-2
 3110 REMF
 3120 DATA-1,0,0,3,0,3,7,8,7,8,10,3,10,3
,13,9,13,9,16,0,16,0,0,-2
 3130 REMG
 3140 DATA-1,0,3,0,13,3,16,6,16,9,13,9,1
1,6,11,6,12,5,13,4,13,3,12,3,4,4,3,5,3,6
,4,6,5,5,5,5,8,9,8,9,3,6,0,3,0,0,3,-2
 3150 REMH
 3160  DATA-1,0,0,0,16,3,16,3,10,6,10,6,
16,9,16,9,0,6,0,6,7,3,7,3,0,0,0,-2
 3170 REMI
 3180 DATA-1,0,0,9,0,9,3,6,3,6,13,9,13,9
,16,0,16,0,13,3,13,3,3,0,3,0,0,-2
 3190 REMJ
 3200 DATA-1,0,3,0,6,3,6,3,4,4,3,5,3,6,4
,6,5,6,16,9,16,9,3,6,0,3,0,0,3,-2
 3210 REMK
 3220 DATA-1,0,0,0,16,3,16,3,11,6,16,9,1
6,5,8,9,0,6,0,3,5,3,0,0,0,-2
```

```
 3230 REML
 3240 DATA-1,0,0,0,16,3,16,3,3,9,3,9,0,0
,0,-2
 3250 REMM
 3260 DATA-1,0,0,0,16,3,16,4.5,12,6,16,9
,16,9,0,6,0,6,9,4.5,7,3,9,3,0,0,0,-2
 3270 REMN
 3280 DATA-1,0,0,3,0,3,8,6,0,9,0,9,16,6,
16,6,8,3,16,0,16,0,0,-2
 3290 REMO
 3300 DATA-1,3,0,0,3,0,13,3,16,6,16,9,13
,9,3,6,0,3,0,-1,3,4,3,12,4,13,5,13,6,12,
6,4,5,3,4,3,3,4,-2
 3310 REMP
 3320 DATA-1,0,0,0,16,6,16,9,13,9,10,6,7
,3,7,3,0,0,0,-1,3,10,3,13,5,13,6,12,6,11
,5,10,3,10,-2
 3330 REMQ
 3340 DATA-1,3,0,0,3,0,13,3,16,6,16,9,13
,9,3,8,2,9,1,8,0,7,1,6,0,3,0,-1,3,4,3,12
,4,13,5,13,6,12,6,4,5,5,4,4,5,3,4,3,3,4,
-2
 3350 REMR
 3360 DATA-1,0,0,0,16,6,16,9,13,9,10,6,7
,9,0,6,0,3,7,3,0,0,0,-1,3,10,3,13,5,13,6
,12,6,11,5,10,3,10,-2
 3370 REMS
 3380 DATA-1,0,0,6,0,9,3,9,7,6,10,4,10,3
,11,3,12,4,13,9,13,9,16,3,16,0,14,0,10,3
,7,5,7,6,6,6,4,5,3,0,3,0,0,-2
 3390 REMT
 3400 DATA-1,0,16,9,16,9,13,6,13,6,0,3,0
,3,13,0,13,0,16,-2
 3410 REMU
 3420 DATA-1,0,16,0,3,3,0,6,0,9,3,9,16,6
,16,6,4,5,3,4,3,3,4,3,16,0,16,-2
 3430 REMV
 3440 DATA-1,0,16,3,0,6,0,9,16,6,16,5,5,
3,16,0,16,-2
 3450 REMW
 3460 DATA-1,0,16,0,0,3,0,4.5,4,6,0,9,0,
9,16,6,16,6,7,4.5,9,3,7,3,16,0,16,-2
 3470 REMX
 3480 DATA-1,0,0,3,8,0,16,3,16,4.5,12,6,
16,9,16,6,8,9,0,6,0,4.5,4,3,0,0,0,-2
 3490 REMY
 3500 DATA-1,0,16,3,16,4.5,8,6,16,9,16,6
,5,6,0,3,0,3,5,0,16,-2
 3510 REMZ
 3520 DATA-1,0,16,9,16,9,12,3,3,9,3,9,0,
0,0,0,4,6,13,0,13,0,16,-2
 3530 REM-
 3540 DATA-1,1,9,8,9,8,6,1,6,1,9,-2
 3550 REM!
 3560 DATA-1,3,16,6,16,6,4,3,4,3,16,-1,3
,0,3,3,6,3,6,0,3,0,-2
 3570 REM" "
 3580 DATA-1,0,0,-2
 3590 REMend
 3600 DATA-2
```

Initialising ROM Images

David Holton discusses the problems of loading and initialising ROM images from boot files, and offers a number of solutions.

The following discussion is based upon my experiences with a Master 128 and with a B+, but in principle applies to all machines to which sideways RAM has been fitted. Note, too, that on a Master ROM slots 4, 5, 6, and 7 may be referred to as 'W', 'X', 'Y', and 'Z' when loading sideways ROM images.

There have been a number of letters and articles in various magazines concerning the initialisation of ROM images. Theoretically, it is necessary to perform a hard Break to get the MOS to recognize the presence of ROM images and to initialise them. Real ROMs are initialised on power-up, but ROM images in RAM are by their nature not in memory until after the power is on. The snag appears when you try to start up such a program from a !BOOT file. You can put:

```
    *SRLOAD Rom_image 8000 W Q
```

in the !BOOT file, and *Rom_image* will load, but not work. On a Master 128, *ROMS will show it, but *HELP won't (incidentally, this is a handy way of finding out whether a ROM image is initialised or not). If you perform Ctrl-Break to initialise the image, you lose anything else set up by the !BOOT file, such as mode changes and function key settings. Press Shift-Break to reboot the !BOOT file, and the ROM image is loaded again, which de-initialises it. Back to square one! If only the image were not de-initialised when loaded a second time, we could just boot a simple !BOOT file twice, pressing Ctrl-Break inbetween.

The MOS does two things when initialising a ROM/RAM image (it can't tell the difference between the two, and treats them exactly the same - from now on, unless explicitly stated, I shall use the word ROM to mean either). Firstly, it reads the type-byte from the seventh byte of each ROM (at &8006), and copies it into a vector-table in RAM which begins at &2A1 (673 decimal). Each ROM's type-byte is held at:

    [&2A1 + the number of the ROM slot]

During normal running, as opposed to start-up, the presence of a type-byte in the table tells the MOS that a ROM is in the slot, and the value of the byte tells it (mainly) whether or not the ROM has a language entry-point.

The other thing that happens on initialisation is that the ROM has a chance to claim any workspace it may need in RAM. Many ROMs do not use anything like all the 16K bytes allocated, and if programmers don't mind their programs being restricted to sideways RAM only, they can use workspace inside the same 16K segment of sideways RAM which contains the program.

Unfortunately, though, you can't write to a real ROM, and so if you want your code to be capable of being loaded from disc or blown into an EPROM without any changes, or if you're writing a commercial ROM, any sideways program needing workspace must use RAM in main memory or in RAM elsewhere (e.g. "hidden RAM"). Programs can be set up to *claim* such workspace on initialisation. A ROM may also need to change one or more vectors in the main RAM on start up. Still, there are many programs which do not need to claim workspace or to redirect vectors, or at least not on start up.

Now we are getting somewhere. Any ROM which doesn't need to claim workspace, or to redirect vectors, can be initialised merely by inserting its type-byte into the table at (&2A1 + slot-number). On a Master, our example *Rom_image* will have its type-byte at 677, since "W" on the Master means "4", and &2A1 + 4 = &2A5, which is 677. The type-byte is usually either 130 or 194 (see table 2). So, writing:

```
    *BASIC
    *SRLOAD Rom_image 8000 W Q
    ?677 = 130 (or 194)
```

in the !BOOT file will set the whole thing up for us, provided that *Rom_image* is the 'easy' sort which doesn't need to claim workspace. This is all that the famous "I" parameter on the Compact does, which is why it only works with some software. Note, however, that on a B+ "W" indicates 12, not 4, so that the vector would be at 685 - see table 1.

Slot	M 128	Vec.Add.	Type	B+128
15/&F	TERMINAL	688 / &2B0	194	Basic
14/&E	VIEW	687 / &2AF	194	
13/&D	ADFS	686 / &2AE	130	SRAM X
12/&C	BASIC	685 / &2AD	96	SRAM W
11/&B	Edit	684 / &2AC	194	
10/&A	ViewSheet	683 / &2AB	194	
9	DFS	682 / &2AA	130	
8	Free socket	681 / &2A9		
7	SRAM Z	680 / &2A8		
6	SRAM Y	679 / &2A7		
5	SRAM X	678 / &2A6		
4	SRAM W	677 / &2A5		
3	Cartridge	676 / &2A4		
2	Cartridge	675 / &2A3		
1	Cartridge	674 / &2A2		SRAM Z
0	Cartridge	673 / &2A1		SRAM Y

Table 1. Slots and vectors on a B+128 & Master 128

How do we know the value of the type-byte? Load in your ROM image, initialise it by Ctrl-Break, and peek at the type-byte using the following routine, which reveals all:

```
10 FOR slot = 15 TO 0 STEP -1
20 vecadd = slot + 673
30 PRINT ~slot;" ";vecadd;" ";?vecadd
40 NEXT slot
```

RUN this, then do *ROMS. You now have the type-byte and vector address of your ROM image.

How do we know whether a given ROM image needs to claim workspace or set vectors? The easy way is to suck it and see! *SRLOAD it, *don't* initialise it by Ctrl-Break, but do so by a poke as above, and give it a thorough trial. Use an unwanted back-up of your disc, though! I've had crashes caused by this that reduce discs to software-spaghetti.

Suppose our ROM is the 'awkward' sort which needs a hard Break. Well, we are now in a position to write a Basic !BOOT file that will *know*, by peeking addresses in the table, whether there is a ROM initialised in a given slot. Write, in a boot file:

```
IF ?677=0 THEN
   OSCLI "SRLOAD Awkward_ROM 8000 W Q"
```

You will need to boot the disc twice - the first time, the ROM image will be loaded but not initialised. Then press Ctrl-Break. The image is now initialised, and the contents of address 677 are no longer zero. Boot the disc a second time, and the image won't be loaded again, so it will stay initialised; you've also got all your function key settings, etc intact. It's very quick, and we've only used one file out of the precious 31 which is all that we ADFS-haters get to use.

There's a bonus here, too. There may be a case when you don't always want to use the ROM. I keep an 'awkward' on-screen clock program as a ROM image on my View discs for my Master. I load it up as above; if - as normally - I don't want the clock, then a normal boot-up loads it, but doesn't set it going. If I'm short of time and need to keep my nose to the grindstone, then booting twice produces the on-screen clock (as a similar conditional line in the !BOOT file also turns it on).

The fact that the ROM identities W,X,Y and Z correspond to different slots on the B+ and the Master can also be a minor complication. For example, my View discs also go into my wife's B+128. We need to let the !BOOT file know which machine it's in: on the B+, slot 14 is empty so ?687 gives 0. On the Master, slot 14 has View in it, so ?687 is always 194. My trusty sideways ROM printer buffer, loaded by the !BOOT file into "W", ends up in slot 12 of the B+ (vector address 685), and slot 4 of the Master (vector address 677). So:

```
IF ?687=0 THEN
   ?685=130 ELSE ?677=130
```

initialises the right slot on each machine - the buffer being the 'easy' sort of program.

The on-screen clock, being the 'awkward' sort, needs a proper Ctrl-Break to initialise it. It is also useless on anything but a Master 128 (which has a built-in real-time clock), so:

```
IF ?687=194 AND ?680=0 THEN
   OSCLI "SRLOAD ROMKLOK 8000 Z Q"
```

loads it only into the Master, slot 7 - and only the first time !BOOT is EXECed, whilst peek 680 is still 0. I only use the second Ctrl-Break to get it going when I want it, as described above. When I do boot it twice, however, a simple *SRLOAD would cause my buffer ROM to be loaded again, too. This would be OK, as the next line will always re-initialise it, but I might as well save loading time and wear on my drive by making it, too, load only the first time:

```
IF ?677=0 THEN
   OSCLI "SRLOAD Buffer 8000 W Q"
```

All we have to do now is poke the correct byte of the vector-table to set up the buffer the first time round, as above:

```
IF ?687=0 THEN
   ?685=130 ELSE ?677=130
```

Note that this is not conditional on 677 or 685 being "empty" before poking them. It doesn't matter - this line takes no significant time to execute, and to add tests would produce a ghastly tangle:

```
IF ?687=0 AND ?685=0 THEN
   ?685=130 ELSE IF ?677=0 ?677=130
```

What a mess! Forget it. The complete !BOOT file is therefore:

```
*BASIC
IF ?687=194 AND ?680=0 THEN
   OSCLI "SRLOAD ROMKLOK 8000 Z Q"
IF ?677=0 THEN
   OSCLI"SRLOAD Buffer 8000 W Q"
IF ?687=0 THEN
   ?685=130 ELSE ?677=130
*WORD etc etc
```

This works on our own two machines; there is no guarantee that these particular numbers will do so on any other Beeb - it all depends what ROMs, Sideways RAM boards and other ironmongery are fitted - especially to a B or B+. The principles, however, can always be implemented.

Bit	Value	Meaning
7	128	ROM has a service-entry point (compulsory).
6	64	ROM has a language-entry point.
5	32	ROM has a 2nd processor relocation address.
4	16	Electron only - soft key expansion.
3	8	ROM is in Z80 code (for Z80 co-processor).
2	4	ROM is in other co-processor code.
1	2	Always set if slot is not empty.
0	1	Never set.

Table 2. Type-bytes in bits

Tip one - do NOT use the same filename for the ROM image file on the disc as any star command needed to use the image once installed. For instance, my clock routine is enabled by *CLOCK. If the file on the disc were called CLOCK, and if I did *CLOCK when the image was not initialised or loaded, the MOS would pass the unrecognised call to the DFS which would load and try to run the program in normal RAM space. The result is a crash; I have therefore named the disc file ROMKLOK.

Tip two - using such a !BOOT file on a Master becomes easier still if your machine is *CONFIGUREd BOOT, as opposed to NO BOOT. The first time, !BOOT runs without any key-press; the second time, the Ctrl-Break also does the boot-up. I much prefer it anyway.

A "normal" ROM for an unexpanded machine will have bits 7 and 2 set (128+2=130), and a "language" ROM will also have bit 6 set (128+64+2=194). If you set a bank of Sideways RAM as *SRDATA or *SRROM, a peek will reveal that only bit 1 is set (the Basic ROM is unique in having no service-entry point, and bit 1 of its type-byte is also reset. Bits 6 and 5 are set, giving 64+32=96).

Special Offers to BEEBUG Members

As explained in the Editorial, members will still receive a discount on our own software for the Archimedes. Below is a list of this software showing members prices inclusive of VAT.

Code	Product	Members Price inc.VAT	Code	Product	Members Price inc.VAT
1407A	ASTAAD3 - 5" Disc (DFS)	9.95	0077B	C - Stand Alone Generator	14.25
1408A	ASTAAD3 - 3.5" Disc (ADFS)	9.95	0088C	Masterfile ADFS BBC 40 T	16.50
1404A	Beebug Applics I - 5" Disc	5.75	0089C	Masterfile ADFS BBC 80 T	16.50
1409A	Beebug Applics I - 3.5"Disc	5.75	0081C	Masterfile ADFS M128 80 T	16.50
1411A	Beebug Applics II - 5" Disc	5.75	0024C	Masterfile DFS 40 T	16.50
1412A	Beebug Applics II - 3.5" Disc	5.75	0025C	Masterfile DFS 80 T	16.50
1421B	Beebug Binder	3.99	0085C	Printwise 40 Track	22.50
1600A	Beebug magazine disc	4.75	0086C	Printwise 80 Track	22.50
1405A	Beebug Utilities - 5" Disc	5.75	0009C	Studio 8	16.50
1413A	Beebug Utilities - 3.5" Disc	5.75			
1450A	EdiKit 40/80 Track	5.75	0074C	Beebug C 40 Track	44.25
1451A	EdiKit EPROM	7.75	0075C	Beebug C 80 Track	44.25
1452A	EdiKit 3.5"	5.75	0084C	Command	29.25
0005B	Magscan Vol.1 - 8 40 Track	12.50	0073C	Command(Hayes compatible)	29.25
0006B	Magscan Vol.1 - 8 80 Track	12.50	0053C	Dumpmaster II	23.25
0011A	Magscan Update 40 track	4.75	0004C	Exmon II	24.00
0010A	Magscan Update 80 track	4.75	0087C	Master ROM	29.25

OTHER MEMBERS OFFERS

Wherever possible we will attempt to include details here of items upon which we have negotiated a special deal. These offers will only be for a limited period while stocks are available. This month these are:

C Programming Language & Stand Alone Generator

Normal Members price £58.50 (inc VAT)

Offer price £39 (inc VAT)

The complete Beebug C Programming Language conforming to the Kernighan and Ritchie standards including the Stand Alone Generator.

Stock code 0082C - 40 track
Stock code 0083C - 80 track

Computer Concepts Mega 3

Normal Members price £87.40 (inc VAT)

Offer price £58.65 (inc VAT)

A complete Wordprocessor/Spreadsheet/Graphics Package incorporating the well known Inter-Word, Inter-Sheet and Inter-Chart.

Stock Code 1119D

5 Games for the Master Compact

Normal Members price £58.50 (inc VAT)

Offer price £5 (inc VAT)

A collection of 5 quality games, well known titles, for the Master Compact on 3.5" disc.

Stock code 1045D

Play it Again Sam 13

Normal Members price £11.35 (inc VAT)

Offer price £7.25 (inc VAT)

Incorporating the following games:

Barbarian II
Hyperball
Percy Penguin
Pandemonium

Stock code 1092B

BEEBUG Survey
Databases

This is the second of our surveys, and has been compiled in the same vein as the survey of word processors in Vol. 9 Nos. 3 & 4. Once again, authors who are familiar with one of the major packages available for the BBC micro have been invited to describe its main features and explain why they like to use it.

Over the years there has been a large number of databases marketed for the Beeb. Some have fallen by the wayside, but a few have soldiered on, and like word processors, each has its own dedicated band of followers. In this issue we will look at *Viewstore*, *Masterfile* and *System Delta*, while next month *Interbase* and *Betabase* will fall under the spotlight.

One point to bear in mind when choosing a database is to consider whether the data can be exported in a standard format recognisable by other databases - CSV (comma separated values) format is the most common. If not, you may find you are locked in to a particular database, particularly if you have a large amount of valuable data stored.

All the databases covered in this issue are available from BEEBUG. All prices quoted are BEEBUG retail prices and include VAT.

ViewStore

Peter Rochford describes Acornsoft's powerful package.

ViewStore (Acornsoft) £42.49

Way back in BEEBUG Vol.4 No.6 I wrote a somewhat enthusiastic review of Acorn's ViewStore ROM-based database. At that time there were a number of database packages already around, most of which I had used to a greater or lesser extent. But with the arrival of ViewStore, here was a totally new concept in database software for the BBC micro, unlike anything else and vastly more powerful.

ViewStore was written by Mark Colton, who was responsible for the rest of the View family of ROM-based productivity software. It shares the same 'command' and 'edit' modes that are a feature of all these packages, and has the same general look and feel. It works with all the BBC filing systems and can operate in any of the Beeb's screen modes. Along with this, it enjoys the ability to work with shadow RAM and can take advantage of a second processor to allow more memory and an extra turn of speed. Among the features that set ViewStore apart are its amazing maximum capacities in terms of file size (4096 megabytes!), record size (60K), number of fields (254) and field size (239). Couple all of this with the ability to display records in both a spreadsheet-type format and the usual card index layout, and you can see why ViewStore is such a potent and flexible piece of software.

This spreadsheet-type display, whereby you can show up to 27 records on screen at once, is one of the most appealing features of ViewStore. It is particularly useful for stock control applications, where it is an advantage to be able to view a large number of stock item records at any one time.

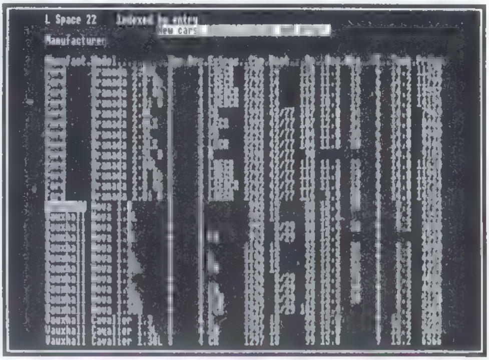
When you set up a database in ViewStore, two main files are created. These are the data file and the format file; the latter contains information on the screen layout. The advantage of this is that you can have as many format files as you like coupled with one data file, allowing great flexibility. Besides these files, a ViewStore database will usually encompass index files as well, for the purpose of searching and sorting on certain fields. You can have up to nine automatically updated indexes and as many manually updated indexes as you require. However, the more auto indexes you have running, the slower the operation of the database becomes. Indeed, with large databases, it can all become quite painfully slow. Even slower is the rebuilding of manually updated indexes, which with a large database can be a dreadfully drawn out affair. I have on occasion re-indexed a large database and decided to pop out to the shops whilst the software was getting on with it!

With many database packages, once you have set up the layout along with the number of records and field sizes etc, you are stuck with it for good. However, ViewStore has a disc-based utility called *Convert* which enables you to change the size of the database and the fields it uses. You can then create a new format file to go with it containing a new layout structure. This can also be done with a utility called *Select* which creates subsets of a data file on which other operations can be performed. These subsets can in fact be converted into separate data files for the creation of new databases based on the original.

Other disc-based utilities provided with the package are *Label* for creating labels, *Link* for linking numeric data from a file which can then be imported into Viewsheet, and *Report* for creating complex reports from data files.

For personal use, ViewStore has remained to this day my only database package. Even now, I use it on the Archimedes under the emulator for all my database needs. On the Arc it runs pretty well at the same speed as on my Master, except when indexing, which on the Arc is blindingly fast. I have two main uses for a database. Firstly I keep records of my friends, acquaintances and business contacts in one enormous file. My second use is keeping track of my LP collection which exceeds 1200 albums. I also keep records of photographic slides and my collection of video recordings.

I have developed a love-hate relationship with ViewStore over the years. Sometimes it frustrates me because of its slowness. This is even more noticeable now I have an Arc, on which everything seems to happen instantaneously! Also, certain operations with ViewStore can be so long-winded to achieve, and involve typing in so many commands. This is particularly true when you are creating subsets. It can make you really angry when you get half way through typing in the fields you want to work on and then forget the name of one of them. You have to list them and then start all over again! Report generation, too, can be real fun. Things don't always turn out as you intended and I have had some real hair-tearing experiences with this particular activity.



```
L Space 22   Indexed by entry
                New cars
Manufacturer
Vauxhall Cavalier 1.3
Vauxhall Cavalier 1.3
Vauxhall Cavalier 1.3GL
```

ViewStore's spreadsheet - type display

ViewStore is the not the friendliest of packages to work with. Some people I have spoken to have commented that they .find it downright intimidating. This I believe is due to the package relying heavily on typed-in commands rather than offering a menu-driven system of operation. This is highlighted if you do not use the package regularly as you will easily forget the commands, and need to refer to the manual. And talking of the manual, this is the biggest criticism I have of ViewStore. In short, it is pretty awful. Everything is probably in there, as they say, but it isn't all that easy to get at and understand. You might well say that a lot is left to the imagination! Some independent guides to the package have been written and these are a definite advantage, particularly to novice users.

Yet despite all these niggles and shortcomings, I still continue to use ViewStore for my needs. The main attraction I believe is that of the spreadsheet type layout. Having got used to that facility, I can't imagine using a database without it. It is certainly still the most powerful database on the Beeb but definitely not the most friendly. Nevertheless, until someone produces something better for the Arc, I will stick with it.

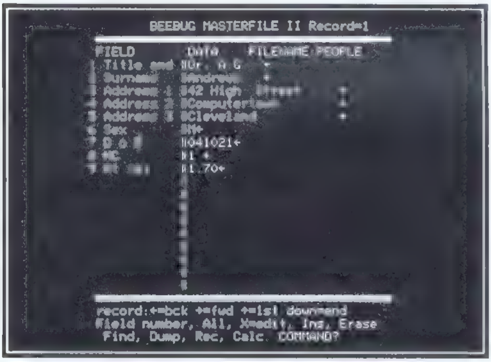
Masterfile

Bernard Hill explains why he uses BEEBUG's Masterfile.

Masterfile (BEEBUG) £22.00 all versions

I'm not sure how long I've been using Masterfile, certainly since my Cassette days (ugh!) and Masterfile I. Masterfile is now in version II and naturally is compatible with the Master, with an ADFS version available. The surprise when examining the Masterfile package is that it is written entirely in Basic and this may reflect its ancient origins. But this in no way detracts from the use of the package, and in fact gives benefits (besides cost) which would not be available in a ROM-based package - more of this later.

But perhaps I had better begin by saying what Masterfile is and is not, and I would add that all my comments explicitly relate to the disc versions.



```
BEEBUG MASTERFILE II Record=1
FIELD          DATA       FILENAME PEOPLE
1 Title and    Dr. A G
2 Surname      Andrews
3 Address 1    42 High Street
4 Address 2    Computertown
5 Address 3    Cleveland
6 Sex          M
7 D o B        041021
8 [illegible]  1
9 [illegible]  1.70
record:←=bck →=fwd ↑=1st down=end
Field number, All, X=edit, Ins, Erase
 Find, Dump, Rec, Calc  COMMAND?
```

Displaying a record in Masterfile (model B version)

The manual claims that Masterfile is a general purpose file management program. I would add that it only handles what are known in database terminology as 'flat-file' applications. In other words it does not easily handle databases which are multi-file and in which one file may cross-reference another, such as an invoice file containing a customer number which must then be looked up in another file to find the customer address. Masterfile does not preclude you writing this sort of add-on application for yourself, however, as all the data structures used in the database are documented in the manual. There is even a sample program to enable you to read database items from Basic. From here, it's only a small step to the production of graphics programs for displaying data in pie charts or other formats, but you will have to write them yourself as there is no graphics support in Masterfile.

However, even if you have no intention of writing your own routines the package is a very powerful database tool at an extremely reasonable cost, and has handled all my personal database needs for the last six years.

Having specified a Database Name after booting up Masterfile, this will be used as the filename of the principal data file. Other files such as index, descriptor and form design files which are associated with the same database will have the same name but reside in other directories. As an example, a sample database called PEOPLE is provided with the package, and files with the same name exist in directories D,E,F,G and 9. (There are slight differences here in the ADFS version but the principle is the same, and instructions are included for using the package with a hard disc). On the DFS version the seven Basic programs which drive the package also reside in the root directory, making a disc catalogue look unnecessarily busy. However if you take the advice of the manual and copy the program files to an empty disc then you can use one disc per database, and with double-sided drives you have the opportunity to use the lower surface for a back-up copy of the data. But it's equally easy to use drive 1 for the database and leave the programs on drive 0, giving maximum disc space allocation for your database.

Incidentally, writing the package in Basic has brought another benefit for the user. The manual gives full details of how to customise the loading program so that you automatically load up a database and set other options after booting, making the one-database-per-disc concept even more attractive.

Having defined a name, the next action is to define the record structure. Up to 18 fields are allowed, and can be of type String, Integer, Date, and Floating-point or Fixed-point real numbers. In actual fact all fields are stored in the database as strings (though you may change the field type later) and this may lead to a waste of disc space, but a DFS disc will still hold a couple of thousand 80-byte records which should really be plenty for a package of this type. Although stored as strings, all validity checking and conversions for sorting purposes are handled invisibly, though the Date-type records are a little inflexible, e.g. for "1 February 1990" you must use "010290". The program will merely swap this internally for "900201" in order to get the sorting order correct. More complex forms such as "01 Feb 90" are not allowed in Date fields, and you can't do date subtractions to find days between dates.

Having defined our database field structure, we need to clear space for the data to prevent the dreaded *Can't Extend* error from occurring. This is a simple process and requires only a projected record count to be given; you can always clear more later.

Now we are ready to enter the data. The screen display shows one record per screen, with field names and lengths as defined previously. You can use the cursor keys to access the next, previous, first and last records, and options are given to jump to a given record number, to edit or amend fields, to dump the current record to printer, or to search for a record. This screen thus forms the principal view of the file and is ideally structured for browsing.

Other options available at the main menu are related to printing, searching and sorting and global file changes. Besides the expected field deletion/addition this last option has some impressive features. Using Basic syntax you could, for example fill a 'total value' field across the whole database with the product of 'price' and 'quantity' fields. Or you could fill a 'VAT' field with 15% of the 'value' field, or even produce a field which holds the cumulative value to date.

The searching and sorting options are fairly comprehensive. It is possible to sort all or part of the file on any field or combination of fields in any combination of ascending or descending orders. Sorts can be actual record movements (*very* slow) or by index files (called tags in this package). A nice feature of the searching options is a 'fuzzy' search which looks for a sub-string in the records. You could extract, print or view all the people in the supplied PEOPLE database who have the title "Dr." and live in a "Lane".

Examination of the print options impresses even more. Apart from the tabular (horizontal) and vertical printing of any selection of fields, there is the ability to print labels and a very impressive *Form Design* option, where a page-per-record output format can be defined. The Form Description Language is rather cumbersome and a long way from WYSIWYG but it is very powerful, even allowing calculations - using any Basic expressions you care to define - which could cut down the size of the database. No need for a separate VAT field if you can calculate it in the form design just for printing! It can even handle conditionals so that in a mail shot "Sir" or "Madam" can be printed depending on the value of a 'sex' field in the database.

Another useful feature of Masterfile is the ability to export data in any format you specify. This means, for example, that if you change your computer you are not condemned to losing all the data you have painstakingly stored over the years.

But in any database package speed is an important factor. A fact of life with the Beeb's filing systems is the miserable speed of PRINT# and BPUT#, and Masterfile cannot overcome this. Sadly it makes no effort to save data in any other way, but the general impression in terms of speed is that the package is adequate, but only just so. If you want to sort 2000 records, do it with an index file and wait at least 10 minutes.

In spite of this however, in conclusion I would like to say that I have remained very happy with my Masterfile package, implementing dozens of databases from Christmas card lists to correspondence chess pairings, saving me hours of work when printing address labels and standard letters. Very good value for money.

An Archimedes version of Masterfile is available, and existing users can upgrade at a cost of £10.

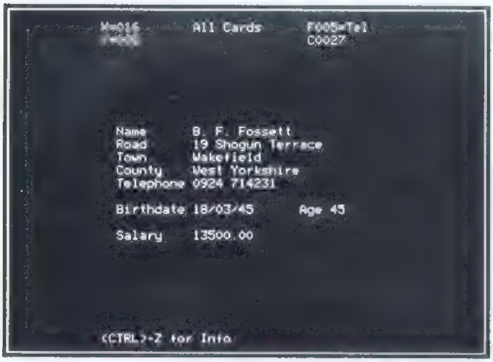
System Delta

Graham Stanley looks at Minerva's System Delta.

System Delta (Minerva) £61.70
System Delta Reference Guide £18.95

In the past six years or so there have been a number of databases released for the BBC Micro and Master 128, ranging from the very basic type to the more sophisticated.

There is a multitude of different uses for a database. Businesses large and small, professionals such as doctors and dentists, even just clubs keep records of all kinds. For example, I know of a Newsagent who keeps details of his paper rounds and customers' addresses on a database. And at home too, you may want to catalogue your video collection or keep a file on friends and other contacts. So a database is potentially a very serious piece of software indeed.



```
     M=016        All Cards       F005=Tel
     [illegible]                  C0027


      Name       B. F. Fossett
      Road       19 Shogun Terrace
      Town       Wakefield
      County     West Yorkshire
      Telephone  0924 714231

      Birthdate  18/03/45      Age 45

      Salary     13500.00



     <CTRL>-Z for Info
```

A card displayed in System Delta

I myself pushed the boat out and went for System Delta from Minerva which seems to cover all my needs. Why? Mainly because I needed a database with many options that could be used according to my specific criteria, and give me the flexibility I wanted.

The System Delta software is supplied as a ROM and a 5.25" floppy disc. Documentation consists of a spiral bound A5 manual which has 84 pages (with index). Once the ROM has been installed you can then insert the disc and boot it, whereupon you will be presented with a title screen which has a number of options.

Selecting B leads to a further multi-option menu screen, which is one of the most important of the System Delta Package. When working with a database, there is also what is called a "quick reference menu" which gives you a number of commands, all accessed by pressing Ctrl together with another key. These allow you to perform functions such as printing a file or adding a record. Some examples of Quick Reference Menu commands are:

`Ctrl-Z`	Info On/Off
`Ctrl-C`	Card Edit
`Ctrl-G`	Go To Field
`Ctrl-V`	Valuate Card
`Ctrl-R`	Remove Card

Before you go ahead and create a file there are numerous points to bear in mind. Firstly you should decide how the format of your database is going to look. I would suggest at this point that you get a pencil and paper, then do a rough layout before you start creating your file.

The next thing you should be looking at is the size of the database. This will obviously vary according to the information you are going to input at the time of creating your file, for example the number and size of records. For instance, a file containing 100 records which will have 100 characters in each record, would require (after taking into account various overheads) approximately 13000 bytes. If you do not think about this before going ahead with your new database file, you could find later that you will very quickly run out of disc space.

Database label generators can often be very confusing to use, but with System Delta nothing could be easier. Suppose that, once you have your file loaded, you would like to run off a mailing list or send a few Christmas cards to your friends. From the main menu select option G and load the format from the disc by entering *Label* and the name of your file. If you now press the Escape key you will see your card index as normal, but then pressing Ctrl-L will display on the screen how your label will look before it is printed. You can choose to print or not to print any field by pressing Ctrl-E. This gives you the chance to make any amendments to your label before printing.

System Delta has a built-in programming language, and I have heard many people say this is difficult to get on with. But it is not that difficult if you have some knowledge of Basic itself, and there is a Reference Guide available which will guide you in the construction of your database.

The programming language is made up of star commands, for example:

**SDadd {File}* add a card to a file.
**SDfile {File}* select a primary file.
**SDdefault* enable/disable default record.

Variables are also used, just as they would be in a Basic program. In the manual supplied with System Delta you will find a few examples of what can be done with the programming system itself. I have to say that the only way to come to grips with this subject is to experiment, and I suggest you do the same, following the examples given in the manual.

Sorting records in System Delta can be achieved either by running a separate Sort program supplied on the disc, or by selecting S from the title screen. When sorting or selecting in System Delta you can of course use wildcards such as the hash symbol to reference any of the fields. As an example, if you want all the names beginning with B, such as Bardwell, Bones, Bramley or whatever, you would use *B#*.

System Delta is very user friendly and has a nice layout, which is easy to follow. The structure of menus and sub-menus makes System Delta very easy to use and enables you to manipulate data exactly as you require.

ADDRESSES

Acorn Computers Ltd,
Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.
Tel. (0223) 245200.

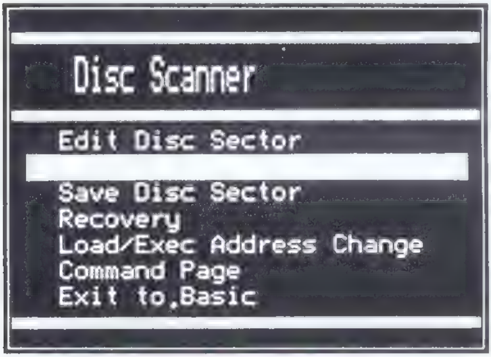
BEEBUG Ltd,
117 Hatfield Road, St Albans AL1 4JS
Tel. (0727) 40303.

Minerva Software,
Minerva House, Baring Crescent, Exeter EX1 1TL.
Tel. (0392) 437756.

ADFS Disc Sector Editor (Part1)

This program by Stefano Spina presents a highly professional approach to a powerful utility for all ADFS disc users.

This comprehensive utility is designed for discs using the ADFS filing system on a Master 128 or Master Compact, though it could be easily adapted for a BBC micro fitted with the ADFS, shadow and sideways RAM. It also offers an extended range of features compared with published in BEEBUG Vol.8 No.3. The program allows any individual disc sector to be loaded into memory, and displayed on the screen in both ASCII code and character formats. At any time you can move on to the next sector or back to the previous one. Each byte of a sector can be edited, and any changes saved back to disc.



```
Disc Scanner
Edit Disc Sector

Save Disc Sector
Recovery
Load/Exec Address Change
Command Page
Exit to Basic
```

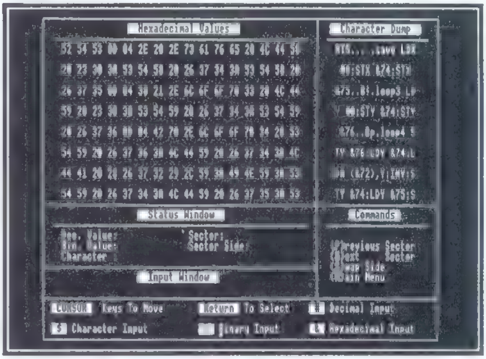
Figure 1. The main menu

The program also provides a comprehensive recovery feature allowing files which have been lost or deleted to be recovered in many instances, a real life saver when this problem occurs.

The complete program is too long to be presented in one issue, and it has therefore been split into two parts. The first part, the still substantial listing given here, provides the main program structure together with an implementation of the first three main menu options. This is a complete working program in its own right. Part two, to be added next month, will implement the remaining functions. Because of the length of the full program the use of shadow RAM is essential.

Type in the program given in listing 1 and save this as *DiscScan*. Because of the length of the eventual program you are advised not to add any additional spaces when entering the program, and this also accounts for the rather dense style of coding employed. When you run the program, you will be asked to put the disc to be examined in drive 0, and then press Return. After selecting this disc as the current drive, the program shows the main menu on screen (see figure 1).

There are seven options in all, the first three of which are active this month, together with the last menu option which terminates the program. Before a disc sector can be examined it must be loaded from disc using the appropriate menu option. The sector address can be given in decimal or hexadecimal (preceded by '&'). Once loaded, you can examine the sector by selecting the first *edit* option in the menu. The main edit screen is then displayed (see figure 2). At any time you can return to the main menu by pressing 'M', or by pressing Escape (which will also abort any other operation).



```
              Hexadecimal Values                    Character Dump
 52 54 53 0D 04 2E 20 2E 73 61 76 65 20 4C 44 58   RTS... .save LDX
 20 23 30 3A 53 54 58 20 26 37 34 3A 53 54 58 20    #0:STX &74:STX
 26 37 35 0D 04 30 21 2E 6C 6F 6F 70 33 20 4C 44   &75..0!.loop3 LD
 59 20 23 30 3A 53 54 59 20 26 37 34 3A 53 54 59   Y #0:STY &74:STY
 20 26 37 36 0D 04 42 70 2E 6C 6F 6F 70 34 20 53    &76..Bp.loop4 S
 54 59 20 26 37 36 3A 4C 44 59 20 26 37 34 3A 4C   TY &76:LDY &74:L
 44 41 20 28 26 37 32 29 2C 59 3A 49 4E 59 3A 53   DA (&72),Y:INY:S
 54 59 20 26 37 34 3A 4C 44 59 20 26 37 35 3A 53   TY &74:LDY &75:S
                 Status Window                        Commands
 Dec. Value:              Sector:
 Bin. Value:              Sector Side:              (P)revious Sector
 Character :                                        (N)ext     Sector
                                                    (S)wap Side
                 Input Window                       (M)ain Menu
 CURSOR Keys To Move       Return To Select         # Decimal Input
 $ Character Input         % Binary Input           & Hexadecimal Input
```

Figure 2. The edit screen

There are three other options in edit mode selected by single key presses: 'N' to move to the next sector, 'P' to move back to the previous sector, and 'S' to change which side of the disc you are examining (referred to as sides 'A' and 'B').

The cursor keys can be used to move around the contents of one sector, and as you do this you will see all the other information on the screen updated at the same time. Pressing Return prompts for a new character to replace that currently highlighted. Input may be as a character, or as a code in binary, decimal or hexadecimal depending on the initial character (all the relevant information is shown on the screen). Once a sector has been edited, return to the main menu and select the third (save sector) option to save the edited sector back to disc.

There are many safeguards built into the program which is much easier to use than might be expected. Remember too that this utility can be used to edit any sector on disc, and that includes the disc directory, as well as the contents of any files. This provides a powerful feature, but one which if misused can also cause as much damage as it can cure. You would be advised to practise with a disc whose contents are unwanted (or which have been backed up using *BACKUP) until you are confident of what you are doing.

A disc sector editor is a powerful tool. You may not use it frequently, but when needed it can literally save hours of work if a file is corrupted, or a disc becomes unreadable.

The remaining disc recovery functions will appear next month together with further help and advice on using this important utility.

Listing 1

```
   10 REM Program DiscScan
   20 REM Version B1.2
   30 REM Author  Stefano Spina
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
   60 :
  100 PROCinit
  110 ON ERROR PROCerr
  120 REPEAT
  130 me%=FNmenu("Disc Scanner",7)
  140 MODE128
  150 ON me% PROCedit,PROCloads,PROCssav
e,PROCrecovery,PROCaddress(0),PROCcomm,P
ROCexit
  160 *FX4,0
  170 UNTIL FALSE
  180 END
  190 :
 1000 DEFPROCalm(M%):LOCAL S$,G%
 1010 PROCstbx(0,28,79,31,0)
 1020 VDU7:PROCin1:PROCcolor(1)
 1030 PRINTTAB(1,29)STRING$(In1%," ")
 1040 IFM%=1 S$="No Sector Loaded"
 1050 IFM%=2 S$="Error in Cyclic Redunda
ncy Check"
 1060 IFM%=3 S$="Sector Not Found"
 1070 IFM%=4 S$="Unrecognised Command"
 1080 IFM%=5 S$="Address Not Allowed"
 1090 IFM%=6 S$="Volume Error"
 1100 IFM%=7 S$="Drive Not Allowed"
 1110 IFM%=8 S$="Unrecognised Error"
 1120 IFM%=9 S$="File Not Found"
 1130 IFM%=10 S$="Input Must Be In Hexad
ecimal (Start With ""&"" Chr)"
 1140 IFM%=11 S$="Illegal Pathname"
 1150 PRINTTAB(FNcnt(S$),29)S$
 1160 PROCcolor(0):S$="Press Any Key"
 1170 PRINTTAB(FNcnt(S$),30)S$:PROCcurs(
0):G%=GET:PROCin1
 1180 ENDPROC
 1190 :
 1200 DEFFNb(V$)
 1210 LOCAL B$,L%:vl%=0:L%=LENV$
 1220 FORF%=1TOL% STEP 1
 1230 B$=MID$(V$,L%-F%+1,1)
 1240 IFB$<>"0"ANDB$<>"1" B$="1"
 1250 vl%=vl%+VAL(B$)*(2^(F%-1))
 1260 NEXT
 1270 =vl%
 1280 :
 1290 DEFFNbin(X%)
 1300 A$="":REPEAT:A$=STR$(X% MOD 2)+A$:
X%=X% DIV 2:UNTILX%<1
 1310 IFLENA$<8 A$="0"+A$:GOTO1310
 1320 =A$
 1330 :
 1340 DEFPROCchg:LOCAL J%:VDU7
 1350 REPEAT
 1360 PRINTTAB(12,26)"Save Changes (Y/N)
":J%=GET:PRINTTAB(4,26)SPC40
 1370 UNTILFNtest(78,J%)ORFNtest(89,J%)
 1380 IFFNtest(89,J%) PROCpv(10,sec%)
 1390 chg%=FALSE
 1400 ENDPROC
 1410 :
 1420 DEFPROCchoose
 1430 LOCAL A%,B%,G%:A%=0:B%=0
 1440 PROCsel(A%,B%)
 1450 PROCflash(A%,B%,FNhex(pk%),chr$)
 1460 IFG%=13 PROCcurs(1):PROCinp(A%,B%)
:PROCsel(A%,B%):PROCcurs(0):GOTO1450
 1470 IFFNtest(77,G%) ELSE GOTO1500
 1480 IFchg% PROCchg
 1490 M%=TRUE:ENDPROC
 1500 IFFNtest(78,G%) ELSE GOTO1540
 1510 IFchg% PROCchg
 1520 sec%=sec%+1:IFsec%>2559 sec%=0
 1530 pg%=0:PROCpv(8,sec%):ENDPROC
 1540 IFFNtest(80,G%) ELSE GOTO1580
 1550 IFchg% PROCchg
 1560 sec%=sec%-1:IFsec%<0 sec%=2559
 1570 pg%=0:PROCpv(8,sec%):ENDPROC
 1580 IFG%=83 pg%=pg%EOR1:ENDPROC
 1590 IFG%=136 A%=A%-1 ELSE IFG%=137 A%=
A%+1
 1600 IFG%=139 B%=B%-1 ELSE IFG%=138 B%=
B%+1
 1610 IFA%<0 A%=15 ELSE IFA%>15 A%=0
 1620 IFB%<0 B%=7 ELSE IFB%>7 B%=0
 1630 PROCsel(A%,B%)
 1640 GOTO1450
 1650 :
 1660 DEFFNcnt(S$)
 1670 LOCAL C%,S%:S%=FNmd
 1680 IFS%=0ORS%=3 C%=80 ELSE IFS%=1ORS%
=4ORS%=6ORS%=7 C%=40 ELSE C%=20
 1690 =INT(C%-LENS$)/2
 1700 :
 1710 DEFPROCcntrl(L%,D%,U%,M%)
 1720 LOCAL B%,C%,I%:B%=FALSE:P%=FALSE
 1730 IFM%=1 M%=3 ELSE IFM%=2 M%=8 ELSE
IFM%=3 M%=2:B%=TRUE
 1740 IFL%>M% PROCreply:P%=TRUE:ENDPROC
 1750 FORI%=1TOL%
 1760 C%=ASC(MID$(V$,I%,1)):IFB% ELSE GO
TO1790
 1770 IFC%<D%ORC%>U% AND C%<65ORC%>70 O%
=TRUE
 1780 GOTO1800
 1790 IFC%<D%ORC%>U% O%=TRUE
 1800 NEXTI%
 1810 IFO% PROCreply:P%=TRUE
 1820 ENDPROC
 1830 :
 1840 DEFPROCcolor(M%)
 1850 LOCAL B%,F%,S%
 1860 S%=FNmd:IFS%=7 ENDPROC
 1870 IFS%=0ORS%=3ORS%=4ORS%=6 B%=129:F%
=1 ELSE IFS%=1ORS%=5 B%=131:F%=3 ELSE B%
=135:F%=7
 1880 IFM%=0 COLOURF%:COLOUR128 ELSE COL
OUR0:COLOURB%
 1890 ENDPROC
 1900 :
 1910 DEFPROCcurs(M%)
 1920 VDU23,1,M%;0;0;0;0;
 1930 ENDPROC
 1940 :
 1950 DEFPROCdiske
 1960 LOCAL X%:X%=?stk%
 1970 IF X%=72 X%=2 ELSE IFX%=80 X%=3 EL
SE IFX%=96 X%=4 ELSE IFX%=97 X%=5 ELSE I
FX%=99 X%=6 ELSE IFX%=101 X%=7 ELSE X%=8
 1980 PROCalm(X%):ENDPROC
```

```
 1990 :
 2000 DEFPROCedit
 2010 LOCAL M%:M%=FALSE:chg%=FALSE
 2020 IFsec%=-1 PROCalm(1):ENDPROC
 2030 PROCscr:REPEAT:PROCout(pg%):PROCch
oose:UNTILM%
 2040 ENDPROC
 2050 :
 2060 DEFPROCerr
 2070 PROCcolor(0):IFERR=17 ENDPROC
 2080 LOCAL S$,G%:*FX4,0
 2090 VDU26:CLS:PROCstbx(0,14,79,19,0):P
ROCstbx(0,17,79,0,1):PROCcolor(1):PRINTT
AB(1,15)STRING$(Inl%," ")
 2100 S$="Error "+STR$ERR+" at line: "+S
TR$ERL:PRINTTAB(FNcnt(S$),15)S$
 2110 PROCcolor(0):VDU28,2,16,78,15:REPO
RT:VDU26:S$="Press Any Key":PRINTTAB(FNc
nt(S$),18)S$:PROCcurs(0):G%=GET
 2120 ENDPROC
 2130 :
 2140 DEFPROCexit
 2150 S$=CHR$129+CHR$157+CHR$135:VDU22,1
35:OSCLI"CLOSE":FORI%=4TO7:OSCLI("SRROM
"+STR$I%):NEXT:PRINTTAB(3,10)S$"    All
Files Closed      "CHR$156:PRINTTAB(3,12
)S$"SideWays Banks Restored   "CHR$156:P
RINT'':END
 2160 :
 2170 DEFPROCflash(A%,B%,A$,B$)
 2180 LOCAL F%,S%:F%=TRUE:*FX4,1
 2190 REPEAT
 2200 IFS% PROCcolor(1) ELSEPROCcolor(0)
 2210 PRINTTAB(3+3*A%,3+2*B%)" "A$" "
 2220 PRINTTAB(58+A%,3+2*B%)B$
 2230 G%=INKEY(40):IFS%=FALSE S%=TRUE EL
SE S%=FALSE
 2240 IFG%=13ORG%=77ORG%=78ORG%=80ORG%=8
3ORG%=136ORG%=137ORG%=138ORG%=139 F%=FAL
SE
 2250 UNTIL NOT F%
 2260 IFG%=13 PROCcolor(1) ELSEPROCcolor
(0)
 2270 PRINTTAB(3+3*A%,3+2*B%)" "A$" ":PR
INTTAB(58+A%,3+2*B%)B$:PROCcolor(0)
 2280 *FX4,0
 2290 ENDPROC
 2300 :
 2310 DEFPROCget(G$,M%)
 2320 LOCAL A$,S$:PROCin1:VDU7
 2330 IFM%=1 S$="Insert the disc to be t
ested into drive 0 - press Return"
 2340 IFM%=2 S$="Change Disc (Y/N)"
 2350 IFM%=3 S$="Replace Disc - press Re
turn"
 2360 PRINTTAB(2,29)S$:PROCcurs(0):REPEA
T:A$=GET$:UNTIL INSTR(G$,A$)<>0
```

```
 2370 PROCin1:get%=ASCA$
 2380 ENDPROC
 2390 :
 2400 DEFFNhex(N%):N$=STR$~N%
 2410 IF LENN$<2 N$="0"+N$
 2420 =N$
 2430 :
 2440 DEFPROCin(A$,B$,C$,D$,E$,A%,B%,C,D)
 2450 LOCAL F$,G$,E%,F%,G%,H%,J%,K%
 2460 C$=FNswap(C$,1):D$=FNswap(D$,1)
 2470 IFFNmd>=0ANDFNmd<=5ANDFNmd<>3 Inr%
=29 ELSE Inr%=22
 2480 IFFNmd=0ORFNmd=3 Inl%=78 ELSE IFFN
md=2ORFNmd=5 Inl%=19 ELSE Inl%=38
 2490 H%=LENA$+3:PROCcolor(0):PRINTTAB(1
,Inr%)A$": "STRING$(A%," ")" "B$
 2500 REPEAT:REPEAT:VDU7:PROCcolor(1):PR
INTTAB(H%,Inr%)STRING$(A%," "):F$="":G%=
0:sr$="":sr=0:sr%=0:K%=0:J%=0:style%=TRU
E:F%=FALSE:REPEAT:E%=FALSE:PROCcurs(1):V
DU31,H%+K%,Inr%:F$=GET$:G%=ASCF$
 2510 IFG%=9 style%=FALSE:GOTO2570
 2520 IFG%=13 THEN GOTO2570
 2530 IFG%=127ANDJ%=0 THEN GOTO2570
 2540 IFG%=127 J%=J%-1:sr$=LEFT$(sr$,J%)
:PRINTTAB(H%+K%,Inr%)CHR$127:K%=K%-1:GOT
O2570
 2550 IFNOTFNin1(C$,F$) THEN GOTO2570
 2560 sr$=sr$+F$:J%=LENsr$:IFJ%>A% E%=TR
UE ELSE K%=K%+1:PRINTTAB(H%-1+K%,Inr%)F$
 2570 REM@in1
 2580 UNTILG%=13ORE%
 2590 IFsr$=""ANDE$="" E%=TRUE
 2600 IFE% THEN GOTO2640
 2610 IFsr$=""ANDE$<>"" sr$=E$:style%=FA
LSE:F%=TRUE
 2620 IFB%>0 IFNOTFNin3 E%=TRUE:GOTO2640
 2630 IFFNin2 ELSE E%=TRUE
 2640 REM@in2
 2650 UNTILNOTE%
 2660 IFF% PROCcolor(0):PROCin1:GOTO2690
 2670 PROCcurs(0):PROCcolor(0):VDU7:PRIN
TTAB(1,Inr%+1)"Confirm (Y/N)"
 2680 REPEAT:G$=GET$:UNTILFNin1("YN",G$)
:IF(ASCG$AND95)=89 PROCin1 ELSE PRINTTAB
(1,Inr%+1)STRING$(13," "):E%=TRUE
 2690 REM@in3
 2700 UNTILNOTE%
 2710 ENDPROC
 2720 :
 2730 DEFPROCin1
 2740 PRINTTAB(1,Inr%)STRING$(Inl%," "):
PRINTTAB(1,Inr%+1)STRING$(Inl%," ")
 2750 ENDPROC
 2760 :
 2770 DEFFNin1(C$,F$)
 2780 IFC$="" =TRUE
```

```
 2790 F$=FNswap(F$,1)
 2800 IFINSTR(C$,F$)>0ANDF$<>"\" =TRUE
 2810 IFLEFT$(C$,1)="\" C$="01"+C$
 2820 LOCAL J%,K%:J%=1:REPEAT:K%=F$>MID$
(C$,J%,1)ANDF$<MID$(C$,J%+1,1):J%=J%+2:U
NTILK%ORJ%>=LENC$ORMID$(C$,J%,1)="\":IFK
%ORF$=RIGHT$(C$,1) =TRUE ELSE=FALSE
 2830 :
 2840 DEFFNin2
 2850 IFD$="" =TRUE
 2860 LOCAL Y$,Z$,W%,X%,Y%,Z%
 2870 W%=LEND$:X%=1:Z%=0:Y$=FNswap(sr$,1)
 2880 REPEAT:Z%=INSTR(D$,"~",X%):Z$=MID$
(D$,X%,(Z%-X%))
 2890 IFZ$=Y$ Y%=TRUE ELSE X%=Z%+1:Y%=FA
LSE
 2900 UNTILY%ORX%>=W%:=Y%
 2910 :
 2920 DEFFNin3
 2930 IFsr$=CHR$13 =TRUE
 2940 LOCAL Z$,X%,Y%,Z%:X%=TRUE
 2950 IFLEFT$(sr$,1)="-" Z$=MID$(sr$,2)
ELSE Z$=sr$
 2960 IFLEFT$(Z$,1)="&" Z%=TRUE:Z$=MID$(
Z$,2) ELSE Z%=FALSE
 2970 IFINSTR(Z$,"&")<>0ORINSTR(Z$,"-")<
>0:=FALSE
 2980 IFLEFT$(Z$,1)="."ANDZ%:=FALSE
 2990 FORY%=65TO70:IFINSTR(Z$,CHR$Y%)<>0
ANDNOTZ%:X%=FALSE
 3000 NEXTY%:IFNOTX%:=FALSE
 3010 sr=EVALsr$:sr%=EVALsr$:IFB%=2 =TRU
E
 3020 IFsr<C OR sr>D =FALSE ELSE =TRUE
 3030 :
 3040 DEFPROCinit
 3050 OSCLI"SHADOW":OSCLI"DIR :0"
 3060 LOCAL S$,F%,G%:VDU22,128
 3070 DIMasm%1000,data%2000:pg%=0:sec%=-
1:Inr%=29:Inl%=78
 3080 FORF%=0TO2STEP2:P%=asm%:[OPT F%:LD
A #114:LDX #(stk% MOD 256):LDY #(stk% DI
V 256):JSR &FFF1:RTS:.stk%:EQUB 0:EQUW d
ata%:EQUW 0:EQUB 8:EQUB 0:EQUB 0:EQUB 0:
EQUB 0:EQUB 0:EQUW 256:EQUW 0:]NEXT
 3090 PROCoption(1):PROCget(CHR$13,1)
 3100 ENDPROC
 3110 :
 3120 DEFPROCinp(A%,B%)
 3130 LOCAL V$,C%,L%,O%,P%,V%
 3140 chg%=FALSE:*FX4,0
 3150 INPUTTAB(4,26)"Input New Value: "V
$:PRINTTAB(4,26)SPC40
 3160 OSCLI"FX4,1":IF V$="" ENDPROC
 3170 PROCcurs(0)
 3180 C%=ASC(LEFT$(V$,1)):IFC%<35ORC%>38
 PROCreply:GOTO3150
 3190 L%=LENV$:V$=RIGHT$(V$,L%-1):L%=L%-
1:O%=FALSE
 3200 ON C%-34 GOTO3210,3230,3240,3250
 3210 PROCcntrl(L%,48,57,1):IFP% THEN GO
TO3150
 3220 IFVALV$>255 PROCreply:GOTO3150 ELS
E V%=VALV$:GOTO3260
 3230 IFL%<>1 PROCreply:GOTO3150 ELSE V%
=ASCV$:GOTO3260
 3240 PROCcntrl(L%,48,49,2):IFP% THEN GO
TO3150 ELSE V%=FNb(V$):GOTO3260
 3250 PROCcntrl(L%,48,57,3):IFP% THEN GO
TO3150 ELSE V%=EVAL("&"+V$)
 3260 PROCpoke(A%,B%,V%,pg%):chg%=TRUE
 3270 ENDPROC
 3280 :
 3290 DEFPROCinput(M%)
 3300 ON M% GOTO3310,3320
 3310 PROCin("Input Sector Number","(000
0/2559  &000/&9FF)","09AF\&","","",4,1,0
,2559):ENDPROC
 3320 PROCin("Input Sector Number","(0/2
559 &000/&9FF)   Return -> Overwrite","0
9AF\&","",CHR$13,4,1,0,2559):ENDPROC
 3330 :
 3340 DEFPROCinv(X%,Y%,A$)
 3350 PROCcolor(1):PRINTTAB(X%,Y%)A$:PRO
Ccolor(0)
 3360 ENDPROC
 3370 :
 3380 DEFFNlen(D%,S$,C$,M%)
 3390 LOCAL L%:L%=LENS$:IFL%>=D% =S$
 3400 IFM%=0 =STRING$(D%-L%,C$)+S$ ELSE
=S$+STRING$(D%-L%,C$)
 3410 :
 3420 DEFPROCloads
 3430 PROCoption(2):PROCinput(1):sec%=sr
%:PROCpv(8,sec%)
 3440 ENDPROC
 3450 :
 3460 DEFFNmd
 3470 LOCAL A%,Z%:A%=&87:Z%=USR(&FFF4)
 3480 =VAL(LEFT$(STR$~(Z%AND&00FF0000),1
))
 3490 :
 3500 DEFFNmenu(H$,O%)
 3510 CLS:VDU22,135:LOCALN$,G%,J%,I%,T%,
P%:T%=(19-O%)DIV2:RESTORE3520
 3520 REM@mn1
 3530 DATA Edit Disc Sector,Load Disc Se
ctor,Save Disc Sector,Recovery,Load/Exec
 Address Change,Command Page,Exit to Bas
ic
 3540 VDU23,1;0;0;0;0;:PRINTTAB(1,T%)CHR
$145CHR$232STRING$(31,CHR$172)CHR$180
 3550 FORJ%=T%+1TOT%+2:PRINTTAB(0,J%)CHR
$145CHR$141CHR$234CHR$131STRING$(29," ")
CHR$145CHR$181:NEXT
 3560 PRINTTAB(1,T%+3)CHR$145CHR$234STRI
NG$(31,CHR$172)CHR$181
 3570 FORJ%=T%+4TOT%+3+O%:PRINTTAB(1,J%)
CHR$145CHR$234CHR$131CHR$156CHR$134STRIN
G$(26," ")CHR$145CHR$156CHR$181:NEXT
 3580 PRINTTAB(1,J%)CHR$145CHR$170STRING
$(31,CHR$172)CHR$37
 3590 FORJ%=T%+1TOT%+2:PRINTTAB(1+(25-LE
N(H$))DIV2,J%)H$:NEXT:FORJ%=1TOO%-1:READ
N$:PRINTTAB(6,J%+T%+3)N$:NEXT
 3600 READN$:PRINTTAB(5,J%+T%+3)CHR$130;
N$
 3610 P%=1:PRINTTAB(4,P%+T%+3)CHR$157CHR
$129
 3620 *FX4,1
 3630 REPEAT:REPEAT:G%=GET:UNTILG%=13ORG
%=138ORG%=139
 3640 IFG%=13 THEN3690
 3650 IFP%<O% PRINTTAB(4,P%+T%+3)CHR$156
CHR$134 ELSEPRINTTAB(4,P%+T%+3)CHR$156CH
R$130
 3660 IFG%=138 P%=P%+1:IFP%>O% P%=1
 3670 IFG%=139 P%=P%-1:IFP%=0 P%=O%
 3680 PRINTTAB(4,P%+T%+3)CHR$157CHR$129
 3690 REM@mn2
 3700 UNTILG%=13
 3710 =P%
 3720 :
 3730 DEFPROCoption(M%)
 3740 LOCAL S$:PROCstbx(0,10,79,12,0):PR
OCstbx(0,28,79,31,0):PROCcolor(1):PRINTT
AB(1,11)STRING$(78," "):VDU7
 3750 IFM%=1 S$="Preset Option"
 3760 IFM%=2 S$="Load Sector Option"
 3770 IFM%=3 S$="Save Sector Option"
 3780 IFM%=4 S$="Recovery Option"
 3790 IFM%=5 S$="Normal Area Recovery Op
tion"
 3800 IFM%=6 S$="Extended Area Recovery
Option"
 3810 IFM%=7 S$="Change Load/Exec Addres
s Option"
 3820 PRINTTAB(FNcnt(S$),11)S$:PROCcolor
(0)
 3830 ENDPROC
 3840 :
 3850 DEFPROCout(N%)
 3860 LOCAL B$,B%,F%,P%:PROCcurs(0)
 3870 FORF%=(N%*128)TO(127+N%*128)
 3880 B%=?(data%+F%):N$=FNhex(B%)
 3890 IFB%>31 B$=CHR$B% ELSE B$="."
 3900 P%=F%-N%*128:X%=P% MOD 16:Y%=P% DI
V 16:PRINTTAB(4+3*X%,3+2*Y%)N$:PRINTTAB(
58+X%,3+2*Y%)B$
 3910 NEXT
 3920 ENDPROC
 3930 :
 3940 DEFPROCpeek(A%,B%,F%)
 3950 pk%=?(data%+16*B%+A%+F%*128)
 3960 ENDPROC
 3970 :
 3980 DEFPROCpoke(A%,B%,C%,E%)
 3990 ?(data%+16*B%+A%+E%*128)=C%
 4000 ENDPROC
 4010 :
 4020 DEFPROCpv(M%,T%)
 4030 LOCAL H%,I%,L%,R%:H%=T% DIV 65536:
R%=T%-H%*65536:I%=R% DIV 256:L%=R% MOD 2
56:?(stk%+6)=H%:?(stk%+7)=I%:?(stk%+8)=L
%:?(stk%+5)=M%:CALLasm%
 4040 IF?stk%<>0 PROCdiske
 4050 ENDPROC
 4060 :
 4070 DEFPROCreply
 4080 LOCAL G%:VDU7:PRINTTAB(12,26)" Ill
egal Input - Press Any Key":PROCcurs(0):
G%=GET:PRINTTAB(4,26)SPC40
 4090 ENDPROC
 4100 :
 4110 DEFPROCscr
 4120 CLS:PROCcurs(0)
 4130 PROCstbx(0,0,78,31,0):PROCstbx(0,1
8,78,0,1):PROCstbx(54,0,0,27,2):PROCstbx
(0,27,78,0,1):PROCstbx(2,2,52,0,1):PROCs
tbx(56,2,76,0,1):PROCstbx(2,20,52,0,1):P
ROCstbx(56,20,76,0,1):PROCstbx(0,24,54,0
,1)
 4140 PROCinv(17,1,"  Hexadecimal Values
  "):PROCinv(57,1,"  Character Dump  "):
PRINTTAB(4,21)"Dec. Value:":PRINTTAB(4,2
2)"Bin. Value:":PRINTTAB(4,23)"Character
 :"
 4150 PRINTTAB(29,21)"Sector:":PRINTTAB(
29,22)"Sector Side:":PROCinv(19,19,"  St
atus Window  ")
 4160 PROCinv(19,25,"  Input Window  "):
PROCinv(61,19," Commands "):PROCinv(2,28
," CURSOR ")::PRINTTAB(12,28)"Keys To Mo
ve":PROCinv(31,28," Return "):PRINTTAB(3
9,28)" To Select":PROCinv(53,28," # "):P
RINTTAB(57,28)"Decimal Input"
 4170 PROCinv(2,30," $ "):PRINTTAB(6,30)
"Character Input":PROCinv(31,30," % "):P
RINTTAB(35,30)"Binary Input":PROCinv(53,
30," & "):PRINTTAB(57,30)"Hexadecimal In
put"
 4180 PRINTTAB(57,22)"(P)revious Sector"
:PRINTTAB(57,23)"(N)ext     Sector":PRIN
TTAB(57,24)"(S)wap Side":PRINTTAB(57,25)
"(M)ain Menu"
 4190 ENDPROC
 4200 :
 4210 DEFPROCsel(A%,B%)
```

```
 4220 PROCpeek(A%,B%,pg%):IFpk%>31 chr$=
CHR$pk% ELSEchr$="."
 4230 PRINTTAB(16,21)STR$pk%"   ":PRINTT
AB(16,22)FNbin(pk%):PRINTTAB(16,23)chr$
 4240 PRINTTAB(42,21);:IFsec%=-1 PRINT"N
one" ELSE str$=FNlen(4,STR$sec%,"0",0):P
RINTFNlen(3,STR$~sec%,"0",0)+"/"+str$
 4250 PRINTTAB(42,22);:IFpg%=0 PRINT"A"
ELSE PRINT"B"
 4260 ENDPROC
 4270 :
 4280 DEFPROCssave
 4290 LOCAL D%,G%,S%:PROCoption(3):IFsec
%=-1 PROCalm(1):ENDPROC
 4300 PROCinput(2):IFsr$=CHR$13 S%=sec%:
GOTO4330 ELSE S%=sr%
 4310 PROCget("YyNn",2):D%=get%
 4320 IFFNtest(89,D%) PROCget(CHR$13,3)
 4330 PROCpv(10,S%)
 4340 IFFNtest(89,D%) PROCget(CHR$13,1)
 4350 ENDPROC
 4360 :
 4370 DEFPROCstbx(A%,B%,C%,D%,M%)
 4380 LOCAL S%,E%,F%:S%=FNmd
 4390 IFS%=3ORS%=6ORS%=7 ENDPROC
 4400 IFS%=1ORS%=4 E%=40 ELSE IFS%=2ORS%
=5 E%=20 ELSE E%=80
 4410 F%=1280/E%:A%=(A%*F%)-1+(F%/2)
 4420 B%=(((31-B%)*32)-1)+16
 4430 IFM%<2 C%=(C%*F%)-1+(F%/2)
 4440 IFM%<>1 D%=(((31-D%)*32)-1)+16
 4450 MOVEA%,B%
 4460 IFM%=0 DRAWC%,B%:DRAWC%,D%:DRAWA%,
D%:DRAWA%,B%:IFS%=0 MOVEA%+2,B%:DRAWA%+2
,D%:MOVEC%+2,B%:DRAWC%+2,D% ELSE IFM%=1
DRAWC%,B% ELSE DRAWA%,D%:IFS%=0 MOVEA%+2
,B%:DRAWA%+2,D%
 4470 ENDPROC
 4480 :
 4490 DEFFNswap(S$,M%)
 4500 IFS$="" =""
 4510 LOCAL A$,F%,H%,I%,L%:L%=LENS$
 4520 FORI%=1TOL%
 4530 H%=ASC(MID$(S$,I%,1)):IFH%<65OR(H%
>90ANDH%<97)ORH%>122 THEN4570
 4540 IF(H%>=65ANDH%<=90) F%=FALSE
 4550 IF(H%>=97ANDH%<=122) F%=TRUE
 4560 IFNOTF%AND(M%=0ORM%=2) H%=H%+32 EL
SE IFF%AND(M%=1ORM%=2) H%=H%-32
 4570 REM@sw1
 4580 A$=A$+CHR$H%
 4590 NEXTI%
 4600 =A$
 4610 :
 4620 DEFFNtest(C%,G%):IF(G%AND95)=C% =T
RUE ELSE =FALSE
```

Control Codes in InterWord and InterSheet

R.G.Sharman details the control codes needed for user programming of the function keys when using InterWord and InterSheet.

The function keys on any BBC micro provide a method for the user to set these up to perform quite complex functions at the press of a single key, and this practice can be extended to situations where commercial ROM software is in use (see BEEBUG Vol.8 No.9 for their use in conjunction with the View word processor). This is possible because commercial software seldom uses all the function key options which are possible (particularly the Shift-Ctrl-fkey combination). Usually, each software function is represented by some code value; the problem lies in trying to represent this in a function key definition, but this is easier than it might at first seem.

FUNCTION KEY DEFINITIONS FOR INTERWORD

InterWord is a powerful and effective word processor. However, if like me you are in the lazy habit of highlighting single lines of text simply by pressing the relevant Shifted function key (for underlining, emboldening etc.) instead of properly marking the line first, you may have noticed that there is a tendency for the highlighting to carry on to the next line as it is typed in, and even backwards to previous lines if they are subsequently amended. This is operator error, not program error, but it is still a nuisance! All is not lost though, because by making full use of control codes and function keys we can carry out incredibly long-winded procedures on InterWord text with only one keypress.

Each of the 52 separate InterWord functions available normally from the keyboard has a single character code within the range 1 - 175 (excluding the printable characters 32-126). These character codes can be represented by control code sequences that will render them usable within function key definitions. Table 1 lists all the actions available in InterWord, the corresponding keyboard equivalent, and the associated control and character codes. Don't be misled by the examples of control codes in the BBC Reference Manual (Part 2 page U6.2) - some of them are wrong!

To replicate the keyboard functions singly or severally, within the function keys, it is only necessary to type the relevant control codes, in the order in which they are to be carried out, into a function key definition. This can be done directly from the InterWord main menu, or more tidily by a suitable !BOOT file which assigns the definitions to the function keys and then calls InterWord. The only other requirement is to issue a *FX228,1 call first to ensure that it is the Ctrl-Shift-fkey combinations which are programmed in this way.

By way of an example, consider underlining a line of text - the normal method would require you to move the cursor to the beginning of the line, insert a marker, move the cursor to the end of the line, insert another marker, move the cursor into the marked area, underline with Shift-f4, and then move the cursor to the end of the line to continue inserting text. This involves 4 Shifted and 3 ordinary keystrokes - the 7 equivalent control codes are:

<table>
<tr><td>|!|L</td><td>move cursor to beginning of line</td></tr>
<tr><td>|!#</td><td>insert marker</td></tr>
<tr><td>|!|M</td><td>move cursor to end of line</td></tr>
<tr><td>|!#</td><td>insert marker</td></tr>
<tr><td>|!,</td><td>back 1 space into marked area</td></tr>
<tr><td>|!|D</td><td>underline marked area</td></tr>
<tr><td>|!|M</td><td>move cursor to end of line to continue editing text</td></tr>
</table>

If these codes are typed exactly as shown to define key (n), i.e.:

```
*KEY n |!|L|!#|!|M|!#|!,|!|D|!|M
```

then when in InterWord editing mode, the single keystroke Ctrl+Shift-fkey(n) will correctly underline the current line of text and leave the cursor positioned ready for further text typing. Not really earth-shattering stuff, but quite a saving on fingertips and key switches, especially if you are writing a document with lots of underlined headings.

There are two practical points to note. The vertical bar character used above (|) appears on the keyboard above '\' to the left of the cursor keys. Its purpose in a function key definition is to represent code values over 127 by adding 128 to the code value of the following character. Thus '|A' represents a code value of 193 (128 + 65).

The power of this facility becomes more apparent when you realise that by the same process, it is possible to call menus, insert embedded commands, alter preferences and so on, all with one selection of a pre-programmed function key - and all ten function keys are available for use with Ctrl-Shift. The only limitation is the amount of memory available for the function key definitions in your machine.

To give another example, the following sequence of codes will underline and embolden a line of text, and send the printer codes for single line double width printing (on an Epson-compatible) - all from one key press!

|!|L	move to beginning of line
|!!	invoke embedded command menu
|!.	move down in the menu to the command line
1,14	commands to printer for single line double width
|[	Escape from the menu
|!#	insert marker
|!|M	move to end of line
|!#	insert marker
|!|L	move to beginning of line
|!|D	underline - Shift-f4
|!|M	move to end of line
|!#	insert marker
|!|L	move to beginning of line
|!#	insert marker
|!|!	embolden line - Shift-f5
|!|M	move to end of line to continue inserting text

InterWord Action	What You Type	Control Code	Character Code
Delete under cursor	|A	Ctrl-A	1
Delete Word	|D	Ctrl-D	4
Goto String prompt	|G	Ctrl-G	7
Tabulator	|I	Ctrl-I or Tab	9
Delete to end of line	|K	Ctrl-K	11
Delete Line	|L	Ctrl-L	12
Return/New line	|M	Ctrl-M	13
Same place next page	|N	Ctrl-N	14
Same place previous page	|P	Ctrl-P	16
Remove markers	|R	Ctrl-R	18
Change case	|S	Ctrl-S	19
Mark whole document	|X	Ctrl-X	24
Mark Word	|Z	Ctrl-Z	26
Escape	|[	Escape	27
Delete before cursor	|?	Delete	127
Underline	|!|D	Shift-f4	132
Bold	|!|E	Shift-f5	133
Dotted Underline	|!|F	Shift-f6	134
Normal	|!|G	Shift-f7	135
Beginning of line	|!|L	Shift←	140
End of line	|!|M	Shift→	141
Bottom of document	|!|N	Shift↓	142
Top of document	|!|O	Shift↑	143
Status Menu (1)	|!|P	Ctrl-f0	144
Preferences Menu	|!|Q	Ctrl-f1	145
Marked Section Menu	|!|R	Ctrl-f2	146
Search & Replace Menu	|!|S	Ctrl-f3	147
Page Layout Menu	|!|T	Ctrl-f4	148
Printer Setup Menu	|!|U	Ctrl f5	149
Control Codes Menu	|!|V	Ctrl-f6	150
Multi File Menu	|!|W	Ctrl-f7	151
Spell Check Menu	|!|X	Ctrl-f8	152
Rom Link Menu	|!|Y	Ctrl-f9	153
Left 1 word	|! \	Ctrl←	156
Right 1 word	|!|]	Ctrl→	157
Down 1 screen	|!|^	Ctrl↓	158
Up 1 screen	|!	Ctrl↑	159
Status menu (2)	|!(sp)	f0	160
Embedded command menu	|!!	f1	161
Insert ruler	|!"	f2	162
Insert marker	|!#	f3	163
Left align text	|!$	f4	164
Centre text	|!	f5	165
Right align text	|!&	f6	166
Justify text	|!'	f7	167
Delete marked region	|!(	f8	168
Copy marked region	|!)	f9	169
Delete under cursor (2)	|!+	Copy	171
Back 1 space	|!,	172	
Forward 1 space	|!-	173	
Down 1 line	|!.	174	
Up 1 line	|!/	175	

Table 1. Control code sequences for InterWord

This achieves the required result by one selection of Ctrl-Shift-fkey instead of no less than 19 (the printer command is 4) various Shifted and normal keystrokes. Obviously not all of the InterWord functions are likely to be suitable or desirable for this treatment, but I'm sure you will find lots of repetitive multi-code sequences that will lend themselves admirably to the function keys.

The procedure is the same whatever the required function:

a. Identify the keyboard actions normally needed.

b. Type the corresponding control codes as a function key definition.

INTERSHEET CONTROL CODES

Continuing the story of control characters in function keys, the equivalent codes for InterSheet are given in table 2. The application of sequences of codes within InterSheet is likely to be less useful than in InterWord, but may still be of help at times. For example, I keep my bank account details on InterSheet, and use the following routine (on a function key) for copying the monthly standing orders and direct debits into the sheet at the beginning of each month:

/C	/copy command
A	denotes Area Copy
A5:E10	sheet area to be copied
|M	Return

It then only remains to type in the top left box of the area to be copied to e.g. A127 followed by Return. This saves all the bother of paging up and down the sheet to find the items required, and eliminates any possibility of "finger trouble". You will probably find several code sequences to enhance your own use of spreadsheets.

InterSheet Action	What You Type	Control Code	Character Code
Set area blank	/B	/B	-
Copy area	/C	/C	
Set number of digits	/D	/D	
Set number format	/F	/F	-
Goto Box	/G	/G	
Hold area at cursor	/H	/H	-
Justify area	/J	/J	-
Lock area	/L	/L	
Change negative sign	/N	/N	
Print	/P	/P	-
Release area	/R	/R	
Unlock area	/U	/U	-
Width of column(s)	/W	/W	-
Zap sheet	/Z	/Z	-
Force recalculation	|I	Tab	9
Return/New line	|M	Return	13
Escape	|[	Escape	27
Current box contents (without ") to command line	|!|K	No kybd equiv	139
Rightmost occupied col.	|!|L	Shift←	140
Leftmost occupied col.	|!|M	Shift→	141
Bottom occupied row	|!|N	Shift↓	142
Top occupied row	|!|O	Shift↑	143
Screen Left	|!|\	Ctrl←	156
Screen Right	|!|]	Ctrl→	157
Screen Down	|!|^	Ctrl↓	158
Screen Up	|!|_	Ctrl↑	159
Edit box at cursor	|!(sp)	f0	160
Change screen mode	|!!	f1	161
Change auto step-on	|!"	f2	162
Justify box	|!#	f3	163
Auto/manual re-calc	|!$	f4	164
Delete box	|!%	f5	165
Insert row	|!&	f6	166
Delete row	|!'	f7	167
Insert column	|!(	f8	168
Delete column	|!)	f9	169
Current box number to command line	|!+	Copy	171
Left 1 box/space	|!,	172	
Right 1 box/space	|!-	173	
Down 1 box/line	|!.	174	
Up 1 box/line	|!/	175	

Table 2. Control code sequences for InterSheet

The numeric code 139, for transferring the contents of the current box (without quotes if it is a label) doesn't appear to have an equivalent function documented in the InterSheet manual - it is a somewhat limited function for which I have yet to find a use - any suggestions?

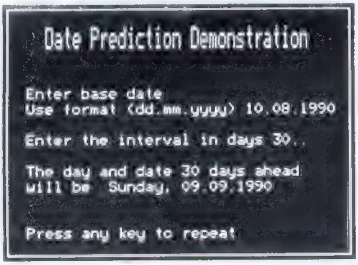
Pinpointing a Date

by Jeff Gorman

INTRODUCTION

The program listed here shows how to pinpoint a future calendar date which lies a given number of days forward from a specified date. The routines to do this can prove useful in a variety of programs.

These routines are presented here as part of a complete working demonstration. Type the program in, save it and then run it to see what it does. You will be asked to enter a reference date, say today's date, and then a number of days from that date. The program will calculate and display the corresponding day of the week and date of the target day. Note that by asking for the year as a four digit number, the program will work correctly with starting dates in this or the next century.



```
Date Prediction Demonstration

Enter base date
Use format (dd.mm.yyyy) 10.08.1990

Enter the interval in days 30

The day and date 30 days ahead
will be  Sunday, 09.09.1990

Press any key to repeat
```

Predicting a date

UNDERSTANDING THE PROGRAM

The routines from lines 1000 to 1240 are concerned with obtaining the necessary data from the user, and converting this into the format required by the date prediction routines which follow. PROCgetDate inputs the reference date using FNgetDate, and the number of days to elapse. The start date is then converted into separate day, month and year numbers (using PROCsplitDate), and these are passed as parameters to FNdayID.

FNdayID is the key function. It counts the number of days that the known date (expressed as day, month and year numbers) is distant from a hypothetical datum of "01.01.1900". This calculated number (*strtDateID%*) is used as an identification number (hence the suffix *ID*). The identifier *Exec* is used in the program to indicate the actual day on which something is due to happen. By adding the value of *daysToExec%* to *strtDateID%* one gets the ID of "exec" day (*execID%*).

In turn PROCreport finds and prints the name of the day, and the date, month and year, represented by *execID%*. A REPEAT loop increments years at yearly intervals as it compares *execID%* with a series of trial IDs for the first day of the year, until the trial ID is greater. The resulting *execYr%* is therefore the preceding year. The figure often given in diaries *execDayNo%*, is found by subtracting the ID for the end of the previous year.

A further REPEAT loop (in FNexecMonth) uses FNmonLen to aggregate the number of days according to the month in question, until *execDayNo%* is greater. The loop count gives the number of the calendar month. As the loop operates it also aggregates the days prior to the terminating value. When, in FNexecDay these are subtracted from *execDayNo%*, *execDate%*, the conventional calendar date is left.

Since the days of the week rotate in an invariable sequence, it is easy to identify the day which an ID represents. The program takes the remainder produced by dividing the ID by seven (*rem%* in FNdayName) to identify the name from a DATA list.

USING THE ROUTINES IN OTHER PROGRAMS

If you want to incorporate the date calculation routines into other programs you will need first to obtain values for *day%*, *mon%* and *yr%*, and then use FNdayID to return *strtDateID%*. Add the interval in days to *strtDateID%* and call PROCreport with this result as a parameter.

The short format "dd.mm.yy" is probably suitable for many purposes, but if the more civilised "ddth, month 19yy" form is required, e.g. "24th April 1990", my article in the May 1989 issue of BEEBUG (Volume 8, No.1) indicates, in principle, how the transformation might be achieved.

FNmonLen is gratefully derived from a BEEBUG *Hints and Tips* item offered by Frank McAree in BEEBUG Vol.5 No.3. Using the formula gives greater speed than the alternative method of repeatedly reading month lengths from DATA statements.

```
   10 REM Program Predict
   20 REM Version B1.4
   30 REM Author  Jeff Gorman
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
   60 :
  100 MODE 7:ON ERROR GOTO 200
  110 Y$=CHR$131:C$=CHR$134:execID%=0
  120 PROCgetData
  130 PROCsplitDate(strtDate$)
  140 strtDateID%=FNdayID(day%,mon%,yr%)

  150 execID%=strtDateID%+daysToExec%
  160 PROCreport(execID%)
  170 PRINT'' C$ "Press any key to repea
t"''':IF GET:CLS:RUN
  180 END
  190 :
  200 PRINT'':REPORT:PRINT" at line ";ER
L
  210 END
  220 :
 1000 DEF PROCgetData:PRINT TAB(0,4);
 1010 FOR A%=1 TO 2:PRINT CHR$141  C$" D
ate Prediction Demonstration":NEXT
 1020 PRINT TAB(0,8) Y$ "Enter" C$ "base
 date"'C$ "Use format (dd.mm.yyyy) .....
.....";
 1030 PRINT TAB(0,11)Y$ "Enter" C$ "the
interval in days ...."
 1040 strtDate$=FNgetDate
 1050 INPUT TAB(28,11) daysToExec%
 1060 IF daysToExec%<0 OR daysToExec%>28
440 PROCmistake
 1070 ENDPROC
 1080 :
 1090 DEF FNgetDate:INPUT TAB(25,9)date$
 1100 ok1=MID$(date$,3,1)="." AND MID$(d
ate$,6,1)="."
 1110 ok2=VAL(MID$(date$,4,2)) > 0 AND V
AL(MID$(date$,4,2)) < 13
 1120 ok3=VAL(LEFT$(date$,2)) > 0 AND VA
L(LEFT$(date$,2)) <= FNmonLen(VAL(MID$(d
ate$,4,2)),VAL(RIGHT$(date$,2)))
 1130 IF NOT (ok1 AND ok2 AND ok3) PROCm
istake
 1140 =date$
 1150 :
 1160 DEF PROCmistake:VDU7
 1170 PRINT'''Y$ "Error - Press any key
to start again"
 1180 IF GET:CLS:PROCgetData:ENDPROC
 1190 :
 1200 DEF PROCsplitDate(date$)
 1210 day%=VAL(LEFT$(date$,2))
 1220 mon%=VAL(MID$(date$,4,2))
 1230 yr%=VAL(MID$(date$,7,4))-1900:ENDP
ROC
 1240 :
```

```
 1250 DEF FNmonLen(mon%,year%)
 1260 =30+ABS((mon%>7)+(mon% AND 1))+(mo
n%=2)*(2+(FNlpYr(year%)))
 1270 :
 1280 DEF FNlpYr(y%):y%=y%+1900
 1290 =(y%>1900 AND y%<2100)AND(y% AND 3
)=0
 1300 :
 1310 DEF FNdayID(days%,mnth%,year%)
 1320 yearDays%=365.25*(year%-1)
 1330 IF mnth%=1:=days%+yearDays%
 1340 monLen%=0:FOR m%=1 TO mnth%-1
 1350 monLen%=monLen%+FNmonLen(m%,year%)
 1360 NEXT:=days%+monLen%+yearDays%
 1370 :
 1380 DEF PROCreport(execID%)
 1390 execDayNo%=FNexecDayID(execID%)
 1400 execMonth$=FNexecMonth(execDayNo%)
 1410 execNme$=FNdayName(execID% MOD 7)
 1420 execDate$=FNexecDay(execDayNo%,exe
cMonth$,execYr%)
 1430 execYr%=execYr%+1900
 1440 execYr$=STR$execYr%
 1450 PRINT' Y$ "The day and date ";days
ToExec%" days ahead"'Y$"will be"
 1460 PRINT TAB(10,14) execNme$", ";exec
Date$;".";execMonth$;".";execYr$
 1470 ENDPROC
 1480 :
 1490 DEF FNexecDayID(execID%):y%=y%-1
 1500 REPEAT:y%=y%+1:a%=FNdayID(01,01,y%
)
 1510 UNTIL a%>execID%:execYr%=y%-1
 1520 =execID%-FNdayID(31,12,execYr%-1)
 1530 :
 1540 DEF FNexecMonth(exDy%):agg%=0:h%=0

 1550 REPEAT:h%=h%+1:pAgg%=agg%
 1560 agg%=agg%+FNmonLen(h%,execYr%)
 1570 UNTIL agg%+1 > exDy%:em$=STR$(h%)
 1580 IF h%<=9 em$="0"+em$
 1590 =em$
 1600 :
 1610 DEF FNdayName(rem%):LOCAL no%,:no%
=-1
 1620 RESTORE 1660:REPEAT:no%=no%+1
 1630 READ prefix$:UNTIL no%=rem% OR no%
=6
 1640 =prefix$+"day"
 1650 :
 1660 DATA Mon,Tues,Wednes,Thurs,Fri,Sat
ur,Sun
 1670 :
 1680 DEF FNexecDay(exDy%,em$,ey%)
 1690 execDate%=execDayNo%-pAgg%
 1700 IF execDate%<=9 execDate$="0"+STR$
execDate% ELSE execDate$=STR$execDate%
 1710 =execDate$
```

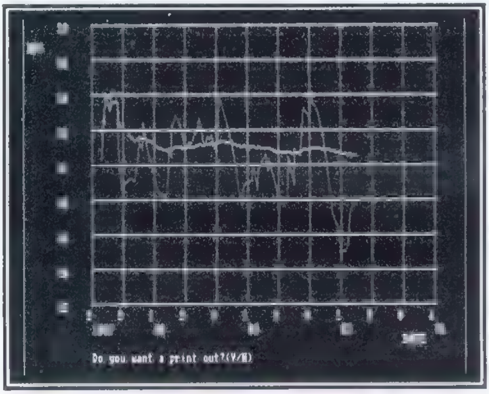
B

Conspicuous Consumption (Part2)

Ralph Maltby concludes his description of a fuel consumption monitor for your car.

Part 1 last month described a program for the Master to tabulate recorded fuel consumption for a car. It also gave some modifications to that listing to suit the model B. In this second part we provide listings for the Master and the model B to extend the program to plot a graph from the computed data (see Figure 1).

The scale of the plot is self-adjusting; the consumption scale, which is in miles per gallon, runs from 75% to 125% of the current value of overall consumption, rounded to the nearest integer. This is usually enough to cover the variations in the short term figures.



Do you want a print out?(Y/N)

A typical screen display produced by the plot routine

Note that the Dump call (line 3390 for the Master & 1170 for the model B) has not been included and that you will have to supply the commands required for your own dump program if you wish to print the graph.

THE MASTER PROGRAM

Listing 3 should be added to Listing 1 to give the complete Master program, ignoring the REM lines at the start. It makes use of the memory released by the shadow RAM to allow ample data to be stored and processed when the complete program is run. The plots are in modes 0 or 1; mode 1 is used for a colour monitor, and mode 0 for a monochrome monitor and for the screen display which is dumped when printing.

THE MODEL B PROGRAM

There is not enough room for an adequate amount of data to be handled easily when the entire application is run as one program. The approach taken here is to have a separate plotting program (Listing 4), which should be saved as FPlot-B. When typing this in, note that three procedures are identical to those in Listing 3 (with the exception of three lines) and have therefore not been listed a second time. Details are given in the REM statements at the end of Listing 4. For ease of typing, the lines can be given the same line numbers as in Listing 3.

When the plot routine is required, the main program stores the calculated data in a separate file called *Temp*, sets PAGE to &1300 and calls FPlot-B.

Unfortunately there is still not quite enough room for modes 0 or 1 to be used, so mode 4 is used instead. If the program is crunched, it might well be possible to use modes 0 and 1 with a reasonably long file.

When the model B plot routine is run, it will first re-load the calculated data from the file *Temp*, and then perform the plot exactly as the Master version.

Listing 3

```
   10 REM        >FConM2
   20 REM Version B1.2 (Master)
   30 REM Author  Ralph Maltby
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
 3260 :
 3270 DEFPROCplot
 3280 VDU19,128,128,0,0,0
 3290 PRINTTAB(10,9)CHR$134"Are you usin
g":PRINTTAB(1,11)CHR$134"a Colour or Mon
ochrome monitor?":PRINTTAB(10,14)CHR$131
"Enter C or M"
 3300 IF FNkey("CcMm") THEN VDU22,1:col$
="C" ELSE VDU22,0:col$="M"
 3310 PROCgrid:PROCplotlines
 3320 MOVE0,-140:PRINT;"Do you want a pr
int out?(Y/N)"
 3330 IF FNkey("YyNn") THEN PROCdump
 3340 VDU22,7:ENDPROC
 3350 :
 3360 DEFPROCdump
 3370 CLS:col$="M":VDU22,0:PROCgrid
 3380 PROCplotlines:VDU29,0;0;2
 3390 REM Insert dump call here
 3400 VDU3:ENDPROC
 3410 :
 3420 DEFPROCgrid
 3430 LOCAL yearorigin%,monthorigin%,yea
rlast%,dateend%,ext,step%,X%,Y%,fuelstep
%,N%
 3440 IFcol$="C":GCOL0,128:VDU19,128,132
,0,0,0 ELSE GCOL0,128:GCOL0,1
 3450 dateorigin%=(VAL(RIGHT$(date$(0),2
))+(VAL(MID$(date$(0),4,2))-1)/12)*10^4
 3460 yearorigin%=dateorigin%DIV10^4
 3470 monthorigin%=VAL(MID$((date$(0)),4
,2))
 3480 yearlast%=VAL(RIGHT$(date$(lastent
ry%),2)):dateend%=(yearlast%+1)*10^4
 3490 ext=(dateend%-dateorigin%)*12/10^4
 3500 IF ext<12 THEN ext=12
 3510 VDU29,200;200;:VDU5
 3520 month%=monthorigin%
 3530 year%=yearorigin%
 3540 MOVE 0,-60:PRINT;(1900+year%)
 3550 MOVE900,-88:PRINT"DATE"
 3560 MOVE-188,750:PRINT;"MPG"
 3570 datescale=1000/ext:
 3580 step%=(ext DIV 12)+1
 3590 FOR X%=0 TO 1000 STEP step%*datesc
ale
 3600 IF month%>12 THEN month%=month%-12
:year%=year%+1:MOVE X%,-60:PRINT;year%
 3610 MOVEX%,0:DRAW X%,800
 3620 MOVE X%-16,-20:PRINT;month%
 3630 month%=month%+step%
 3640 NEXT
 3650 M%=0.75*mpgall%(lastentry%)/100
 3660 m%=M%:fuelstep%=2
 3670 IF 100*M%MOD100>50 THEN M%=M%+1
 3680 N%=1.25*mpgall%(lastentry%)/100
 3690 IF (N%-M%)MOD2>0 THEN N%=N%+1
 3700 scale=800/(N%-M%)
 3710 FOR Y%=0 TO 800 STEP fuelstep%*sca
le
 3720 MOVE 0,Y%:DRAW 1000,Y%
 3730 MOVE -100,Y%+4:PRINT;m%
 3740 m%=m%+fuelstep%:NEXT
 3750 ENDPROC
 3760 :
 3770 DEFPROCplotlines
 3780 GCOL0,1:IF col$="C" THEN VDU19,1,2
,0,0,0
 3790 entry%=1:PROCgriddate(1)
 3800 MOVE (date%-dateorigin%)*datescale
*12/10^4,(mpgall%(entry%)/100-M%)*scale
 3810 entry%=1:REPEAT entry%=entry%+1
 3820 PROCgriddate(entry%)
 3830 IF mile%(entry%)<1050 THEN MOVE (d
ate%-dateorigin%)*12/10^4*datescale,(mpg
all%(entry%)/100-M%)*scale ELSE PLOT21,(
date%-dateorigin%)*datescale*12/10^4,(mp
glast%(entry%)/100-M%)*scale
 3840 UNTIL entry%=lastentry%
 3850 IF col$="C" THEN GCOL0,2:VDU19,2,1
,0,0,0
 3860 entry%=1:PROCgriddate(1)
 3870 MOVE (date%-dateorigin%)*datescale
*12/10^4,(mpgall%(entry%)/100-M%)*scale
 3880 entry%=1:REPEAT entry%=entry%+1
 3890 PROCgriddate(entry%)
 3900 DRAW(date%-dateorigin%)*datescale*
12/10^4,(mpgall%(entry%)/100-M%)*scale
 3910 UNTIL entry%=lastentry%
 3920 MOVE 0,0
 3930 ENDPROC
 3940 :
 3950 DEFPROCgriddate(entry%)
 3960 date%=(VAL(RIGHT$(date$(entry%),2)
)+(VAL(MID$(date$(entry%),4,2))-1)/12 +V
AL(LEFT$(date$(entry%),2))/365)*10^4DIV1
 3970 ENDPROC
```

Listing 4

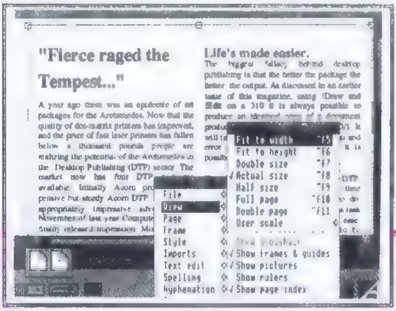
```
   10 REM        >FPlot-B
   20 REM Version B2.2
   30 REM Author  Ralph Maltby
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
   60 :
  100 DIM date$(200),mpgall%(200),mpglas
t%(200),mile%(200)
```

continued on page 46

RISC USER

The Archimedes Magazine & Support Group

Risc User continues to enjoy the largest circulation of any magazine devoted solely to the Archimedes range of computers. Now about to start its fourth year of publication, it provides support for all Archimedes users at work (schools, colleges, universities, industry, government establishments) and home.



"Fierce raged the Tempest..."

A year ago there was an epidemic of art packages for the Archimedes. Now that the quality of dot-matrix printers has improved, and the price of fast laser printers has fallen below a thousand pounds people are realizing the potential of the Archimedes in the Desktop Publishing (DTP) sector. The market now has four DTP available. Initially Acorn pro pensive but sturdy Acorn DTP appropriately impressive adv November of last year Compute Study released Impression. Ma

Life's made easier.
The biggest fallacy behind desktop publishing is that the better the package the better the output. As discussed in an earlier issue of this magazine, using !Draw and !Edit on a 310 it is always possible to produce an identical copy of a document produ

Existing Beebug members, interested in the new range of Acorn micros, may either transfer their membership to the new magazine or extend their subscription to include both magazines.

A joint subscription will enable you to keep completely up-to-date with all innovations and the latest information from Acorn and other suppliers on the complete range of BBC micros. RISC User has a massive amount to offer to enthusiasts and professionals at all levels.

TIGER

A Wimp multi-tasking extended CATalogue which enables you to add descriptive information to files and directories.

ICON BAR CLOCK

A Wimp application which places a combined analogue and digital clock on the icon bar.

INTRODUCING C

A wide ranging new series on C, a major programming language for the Archimedes.

WP/DTP

Anew column which offers hints on using different DTP and WP packages. The first article concentrates on 1st Word Plus

THE BRITISH ISLES IN RELIEF

An entertaining program which creates a coloured map of the British isles.

ASSEMBLER WORKSHOP

A major series for the more advanced ARM processor programmer. The latest one provides an introduction to heap management.

TEMPEST

A review of the latest DTP package for the Archimedes, an enhanced version of Acorn DTP.

MASTERING THE WIMP

A major series for beginners to the Wimp programming environment. The most recent installment is dedicated to the Message system and Object dragging.

INTO THE ARC

A regular series for beginners. The latest article explains expansion cards.

ARCADE

A round-up of the latest games for the Archimedes, including *The Pawn* and *Guild of Thieves* from Magnetic Scrolls, and *Apocalypse* (one of the very best games yet for the Arc) from Fourth Dimension.

As a member of BEEBUG you may extend your subscription to include RISC User for only £8.10 (overseas see below).

Don't delay!
Phone your instructions now on (0727) 40303

Or, send your cheque/postal order to the address below. Please quote your name and membership number. When ordering by Access, Visa or Connect, please quote your card number and the expiry date.

SUBSCRIPTION DETAILS

Destination	Additional Cost
UK,BFPO &Ch Is	£ 8.10
Rest of Europe and Eire	£12.00
Middle East	£14.00
Americas and Africa	£15.00
Elsewhere	£17.00

RISC User, 117 Hatfield Road, St Albans, Herts AL1 4JS, Telephone (0727) 40303, FAX (0727) 60263

BEEBUG WORKSHOP

Searching (Part 2)
Backtracking algorithms

by Bernard Hill

If you attempted the eight queens problem I left you with at the end of last month's article, you will have found how long permutation searching can take. In this article we shall be looking at ways in which we can avoid looking at all the possibilities, and so shorten the search process.

Let's start by examining this problem of placing eight queens on a chessboard so that none is attacking any other - i.e. so that no two lie on any vertical, horizontal or diagonal line.

As mentioned last month, we obviously have only one queen in each column, and each in a different row number so that the positions of the queens can be represented by an array:

```
DIM rownum(8)
```

of the row numbers on which the queens stand, and the numbers in this array will be a permutation of the numbers 1 to 8. The problem of testing for queens on the same diagonal is solved by calculating the sum of each piece's row and column co-ordinates: those on the same NW-SE diagonal will give the same sum, and queens on the same NE-SW diagonal will have the same difference of their co-ordinates. Thus given a permutation of the numbers 1 to 8 in the array 'rownum', we can test for mutual attacks with a function:

```
10000 DEF FNattack
10010 LOCAL col,d,bad
10020 FOR d=2 TO 16:d1used(d)=0:NEXT
10030 FOR d=-7 TO 7:d2used(d)=0:NEXT
10040 col=0
10050 REPEAT
10060   col=col+1
10070   d=rownum(col)+col
10080   bad=d1used(d)
10090   d1used(d)=1
10100 UNTIL bad=1 OR col=8
10110 IF bad THEN =TRUE
10120 col=0
10130 REPEAT
10140   col=col+1
10150   d=rownum(col)-col
10160   bad=d2used(d)
10170   d2used(d)=1
10180 UNTIL bad=1 OR col=8
10190 =bad
```

This function presupposes that we have two arrays 'd1used' and 'd2used' which have been declared initially. Those of you with eagle eyes will have spotted that the index of 'd2used' has to go negative (see line 10030). BBC Basic doesn't allow this, but we could fudge this by declaring it as DIM d2used(14) and adding 7 onto all index calculations. In this case, however, there's a subtler - and faster - solution. If we use indirected arrays, e.g.:

```
DIM d1used 16
```

then we can also use:

```
DIM dummy 14 : d2used=dummy+7
```

and use d2used?x where x can go from -7 to +7!

I leave you to the exercise of putting this function into last week's permutation problems.

A BACKTRACKING ALGORITHM

But if you were to attempt to solve this problem yourself, you would not generate permutations. Logically you might try the following:

1. Place the first queen in square a1.

2. Now the second queen goes in the second column. Obviously not at b1 but equally obviously not at b2 since this is on a diagonal line to a1.

And this is the important part: we can shortcut further placings of the other six queens on the basis of the impossibility of the point we're trying at the moment (in this case b2).

Here is a pseudocode algorithm for implementing such a process. It is called a *backtracking* algorithm because the algorithm automatically ‘backs off’ blind alleys. The key to this is the "unmake move" statement:

```
DEF PROCtry(n)
REPEAT
  select next move
  IF acceptable THEN
     make move
     IF n=max depth THEN print solution
                   ELSE PROCtry(n+1)
     unmake move
UNTIL no more moves
ENDPROC
```

Note that I am using indentation to indicate logical grouping, so that the IF statement covers the next four lines.

If we call the procedure with PROCtry(1) then it will generate all solutions to whatever problem we program. If we are merely searching for any solution (as opposed to all) then of course we can put and END statement after our ‘print solution’ line.

Listing 1 is a solution to the queen's problem along these lines, but I have generalised it to N queens on an NxN chessboard, and the addition of a few lines of graphics means that you can watch the process as it searches. You can see that the vast majority of searches fail well before a queen is placed in the right-hand row.

As written the program will not handle chessboards over 20x20. This is because of the nesting limit of the REPEAT statement: a maximum of 20 REPEAT nestings is allowed by BBC Basic, and the recursion means that we are leaving one REPEAT uncompleted when another starts (a FOR loop is even worse; only 10 are allowed). If you want to search on a larger chessboard then you'll have to replace REPEATs with GOTOs.

Again the program is written for clarity, not speed. To speed up the program you could:

Listing 1

```
   10 REM N Queens problem
   20 REM Version B1.1
   30 REM Author  Bernard Hill
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
   60 :
  100 MODE7
  110 INPUT "How big a chess-board";N
  120 MODE 1
  130 DIM rowused N,d1used 2*N,dum 2*N
  140 d2used=dum+N
  150 FOR i=1 TO N:rowused?i=0:NEXT
  160 FOR i=2 TO 2*N:d1used?i=0:NEXT
  170 FOR i=0 TO 2*N:dum?i=0:NEXT
  180 REM draw board
  190 s=800/N:GCOL 0,1
  200 FOR i=0.5 TO N+.5
  210 MOVE s*i,s/2:PLOT 1,0,N*s
  220 MOVE s/2,s*i:PLOT 1,N*s,0
  230 NEXT
  240 GCOL4,3:VDU5,29,-16;16;
  250 :
  260 T=0
  270 PROCtry(1)
  280 VDU4
  290 PRINT T;" solutions"
  300 END
  310 :
 1000 DEF PROCtry(row)
 1010 LOCAL col:col=0
 1020 REPEAT
 1030 col=col+1
 1040 REM   test for illegal place...
 1050 IF rowused?col THEN 1150
 1060 IF d1used?(row+col) THEN 1150
 1070 IF d2used?(row-col) THEN 1150
 1080 REM   OK so place it
 1090 MOVE row*s,col*s:PRINT "Q";
 1100 rowused?col=1
 1110 d1used?(row+col)=1
 1120 d2used?(row-col)=1
 1130 IF row=N THEN T=T+1:VDU7:z=GET ELS
E PROCtry(row+1)
 1140 rowused?col=0:d1used?(row+col)=0:d
2used?(row-col)=0:MOVE row*s,col*s:PRINT
 "Q";
 1150 UNTIL col=N
 1160 ENDPROC
```

1. Omit the graphics (gains about 30%)
2. Replace all variables by resident integers (another 30%)
3. Replace the GOTO statements in lines 1050-1070 with "UNTIL col=N:ENDPROC" (which is

what they actually implement if the statement is true). About 10%.

Overall result: a gain of 3 times in speed!

KNIGHT'S TOUR

To complement Eric Bramley's Knight's Tour program of Vol. 7 No. 2 which invites you to test your skill, Listing 2 is a chance to tell your Beeb to find a solution for itself. The basic program is very similar to the Queen's problem, all we need is to pre-generate arrays of 8 knight moves A (x-moves) and B (y-moves) (lines 1000-1010) and keep an array H(N,N) for each square which indicates the move number when the square was visited, or 0 if it has not yet been used.

Listing 2

```
  10 REM Knight's tour
  20 REM Version B 1.1
  30 REM Author  Bernard Hill
  40 REM BEEBUG  October 1990
  50 REM Program subject to copyright
  60 :
 100 MODE7
 110 INPUT "Size of board";N
 120 MODE129:REM MODE 4 on Model B
 130 NSQ=N*N
 140 DIM A(8),B(8),H(N,N)
 150 FOR i=1 TO 8:READ A(i),B(i):NEXT
 160 REM plot board
 170 GCOL0,1:s=800 DIV N
 180 FOR i=0.5 TO N+0.5 STEP 1
 190 MOVE s*i,s/2:PLOT 1,0,N*s
 200 MOVE s/2,s*i:PLOT 1,N*s,0
 210 NEXT:GCOL 4,3
 220 :
 230 TIME=0:H(1,1)=1
 240 PROCtry(2,1,1)
 250 REM if you get here there was...
 260 PRINT"No solution"
 270 END
 280 :
1000 DATA 2,1, 1,2, -1,2, -2,1
1010 DATA -2,-1, -1,-2, 1,-2, 2,-1
1020 :
2000 DEF PROCtry(i,x,y)
2010 REM try from sq (x,y) at move i
2020 LOCAL k,u,v:k=0
2030 REM repeat
2040 k=k+1
2050 u=x+A(k):v=y+B(k)
2060 IF u<1 OR u>N THEN 2150
2070 IF v<1 OR v>N THEN 2150
2080 IF H(u,v)>0 THEN 2150
2090 H(u,v)=i:REM make move
2100 MOVE x*s,y*s:PLOT 6,u*s,v*s
2110 IF i=NSQ THEN PRINT "Solution in "
TIME/100" sec.":END
2120 PROCtry(i+1,u,v)
2130 H(u,v)=0:REM undo move
2140 MOVE x*s,y*s:PLOT 6,u*s,v*s
2150 IF (k<8) THEN 2030:REM until k=8
2160 ENDPROC
```

Generating a new move (u,v) from a current position (x,y) is simply a matter of adding the k'th move number to the co-ordinates (line 2050). These new co-ordinates must lie on the board (lines 2060-2070) and be unused (line 2080). Success is indicated by the depth of iteration being N*N.

Note that as mentioned above we have had to replace the REPEAT with GOTO statements as our nesting level would have meant that we can only solve for boards up to 4x4 (16 squares). In any case, the depth of recursion used means that even using shadow RAM (if available) and with PAGE set at &E00 we can only solve boards up to about 16x16 since the depth of recursion then goes up to 256!

However, BE WARNED that the program as given takes about 20 minutes to solve a 5x5 board and about 12 days to find a solution to an 8x8 board! To find a faster solution we need to be more circumspect about the order in which we generate moves. By generating all legal moves first (on entry into PROCtry), and then sorting them so that those nearest the corners are tried first, I generated a solution to an 8x8 board in only a few minutes. I leave that as an exercise for you!

Have some fun watching the program solve the (trivial for humans!) problem of covering the board with king's moves. Just alter the DATA statements in lines 1000-1010.

LAST MONTH'S PROBLEM

Those of you who had a go at the permutations problem in the last issue should have found that there are 9 distinct solutions to the problem. Interestingly, two sets of multipliers give the same product:

```
27*198=5346  and  18*297=5346
42*138-5796  and  12*483=5796
```

Monix: A Machine Code Monitor (Part 2)

by Richard Taylor

This month, the remaining eight utilities are added to the program. Load up your copy of MONIX1 from last month and select mode 7 (the program gets rather long). Then type in this month's listing keeping to the line numbers exactly as listed so that it will merge correctly with part one. Alternatively, if you have BEEBUG's Toolkit then type this part in separately with correct line numbers, save it as 'MONIX2', then load up part one and type:

```
*MERGE MONIX2
```

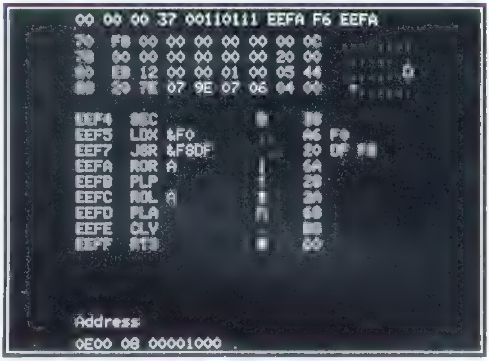
Remember that unwanted routines can be left out (see part one) and you can lower the new value of PAGE by doing this. When you have the full program save it and then run it. Again see part one for details of how to get the assembled program running. The source program should never need to be run again unless you need to make any changes, or the original PAGE value on your machine changes. Normally whenever you require Monix just type *MN.

It would be relatively easy for users to add any machine code utilities of their own which they had written. Any suggestions would be welcome.

PART TWO OPTIONS

E - Edit Memory.
Input the start address. The display will then show the address, the hex value, the binary value (bitwise), and the ASCII character. The zero page can be viewed but not edited. To adjust the user variables (&70-&8F) use option U. For editing use the following keys:

A - move down through memory.
Z - move up through memory.
B - enter new hex value for the address and the edit address will then move on one, enabling quick editing.
S - enter new ASCII character for the address and again the edit address will move on one.
Escape - exit editor.



```
00 00 00 37 00110111 EEFA F6 EEFA
70   F8 00 00 00 00 00 00 0C
78   00 00 00 00 00 00 20 00
80   EB 12 00 00 01 00 05 44
88   20 7E 07 9E 07 06 04 00

EEF4   SEC                38
EEF5   LDX &F0            A6 F0
EEF7   JSR &F8DF          20 DF F8
EEFA   ROR A              6A
EEFB   PLP                28
EEFC   ROL A              2A
EEFD   PLA                68
EEFE   CLV                B8
EEFF   RTS                60

Address
0E00 08 00001000 .
```

Editing Memory from &E00

U - Edit the User Variables (&70-&8F).
Input the single-byte zero page address and then the new value wanted. The value will be shown in the display on the menu screen.

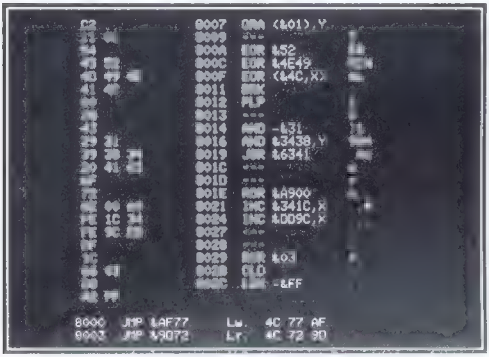
S - String Search.
Enter the string to be searched for, and then the pages to search between (to search between &4100 and &5BFF enter 41 and then 5B). The hex values of the bytes being searched for will then be shown underneath the input. Then the page values currently being searched through will be displayed followed by the addresses of all occurrences of the string in that page (usually none). Look out for 'false' finds, usually in page 03 and anywhere in the monitor itself.

B - Byte Search.
Enter the bytes to be searched for pressing Return after each one. Press Return again when no more bytes are to be entered (the final Return will enter 00 but this will not be searched for). Then follow the same instructions as for the String Search.

J - JSR.
Any subroutine can be accessed with this command. Input the address, followed by the accumulator, X register and Y register values which are required on entering the routine. On

return from the routine the register values will be shown again, allowing use of OSBYTE calls which send back values. This can be very useful for testing whether routines that you have written actually work.

M - Memory Mover.
Enter the start address, the number of bytes to be moved, followed by the destination address. The mover will cope with moves in both directions and overlaps.



```
0007  ORA (&01),Y
0009  ---
000A  EOR &52
000C  EOR &4E49
000F  EOR (&4C,X)
0011  BRK
0012  PLP
0013  ---
0014  AND -&31
0016  AND &3438,Y
0019  JMP &6341
001C  ---
001D  ---
001E  ROR &A900
0021  INC &341C,X
0024  INC &D09C,X
0027  ---
0028  ---
0029  ORA &03
002B  CLD
002C  [illegible] -&FF

8000  JMP &AF77     Lw.  4C 77 AF
8003  JMP &9072     Lr.  4C 72 90
```

Using the disassembler

P - Prompt on Entering On/Off.
When this is turned on, each time the monitor is accessed (from within a program, say) a prompt seeking confirmation will appear in the top left hand corner of the screen. If any key other than 'Y' is pressed, any program will carry on with what it was doing. This also allows a check of where the computer had got to before entering the monitor.

D - Disassembler.
Input an address, and disassembly will start immediately from that address. Be careful, as the first few instructions may be false since disassembly may have started in the middle of an instruction or block of data. If the op-code is not recognised then '—' will be printed. The first column shows the address, which is followed by the assembler command, the ASCII characters of the bytes and then the bytes in hex. The ASCII characters can be useful to show data and therefore reveal false assembly language. These keys are needed:

Shift - move on for the next eight instructions.
Escape - exit Disassembler.

PROGRAM NOTES

OPTION ROUTINES

.edit prints the hex, ASCII and binary values of the byte being looked at. The byte can be changed either by giving it a new hex value or by giving it the value of an ASCII character. The zero page cannot be edited but *.editusr* allows the stored user variables to be changed by asking for the address and new value. The values will be put into &70-&8F when an exit is made from the monitor, as the monitor obviously uses these locations itself.

.bysrch gets an input of up to ten hex bytes from '.slct' and places them in '.srchline'. It then goes to the main search routine at '.bgsrch', which asks for the area to be searched and then moves through this area byte by byte. If the first byte matches then it tries the second and so on until a complete match is made in which case it prints out the address.

.stsrch takes the input string from '.inline' and puts it in '.srchline' before carrying on to '.bgsrch' (see above).

.move goes to '.upmove' or '.downmove' depending on which way through the memory the move is to take place. This is to allow for overlaps. '.upview' starts at the end and works to the beginning, and vice versa for '.downview'

.prompt prints out the necessary message and then toggles the byte at '.prmvar' between 0 and 1 (prompt off and on). See '.monitor' at the beginning of the program.

.jsr asks for the address to go to, and the register values to be sent. They are then printed on return. At the end of the routine, the normal 'JMPkeyrts' cannot be used as it is likely that some of the monitor's zero page variables have been overwritten and they must be reset.

.dis loads up the three bytes at and after the address stored in *memps*. These will not always be needed depending on the instruction. The opcode's addressing mode is found from the data at '.adrmode'. For example 'I' stands for implied addressing mode which is only a one byte instruction (no operands). Then *memps* is incremented by the required amount, in this case one. The assembler mnemonic is printed out from the data at '.mnemstr' together with any operands, followed by the hex bytes (or machine code) and the ASCII values. The disassembler has a counter allowing its use in the menu screen and for it to stop every eight bytes. It also has many subroutines of its own which follow after it.

.t8 labels the start of the part two text.

.mnemstr marks the data for the assembler mnemonics.

.mnemvec gives the mnemonic to be printed for each opcode (0-255). Note that mnemonics like 'LDA' have many different addressing modes and so many different opcodes.

Lastly *.adrmode* gives the addressing mode for each opcode.

```
3000 .edit LDX#(t1-tbs):JSRprtxt
3010 LDA#memps:JSRslctwrd:JSRonw
3020 .bgedit LDX#(t8-tbs):JSRprtxt
3030 JSRprwrd:LDY#0:LDA(memps),Y
3040 JSRprnm:JSRbitwise:JSRasc:JSRonw
3050 JSRrd:TAY:JSRspc:TYA:LDY#0
3060 CMP#ASC"A":BEQdownedit
3070 CMP#ASC"Z":BEQupedit
3080 LDXmemps+1:BEQbgedit
3090 CMP#ASC"B":BEQbyteedit
3100 CMP#ASC"S":BNEbgedit
3110 JSRrd:STA(memps),Y
3120 JMPupedit
3130 .downedit DECmemps:LDAmemps
3140 CMP#255:BNEcseedit:DECmemps+1
3150 .cseedit:JMPbgedit
3160 .upedit INCmemps:LDAmemps
3170 BNEccledit:INCmemps+1
3180 .ccledit JMPbgedit
3190 .byteedit LDA#var:JSRslct
3200 LDY#0:STA(memps),Y:JMPupedit
```

```
3210 :
3220 :
3230 .editusr LDX#(t9-tbs):JSRprtxt
3240 LDA#memps:JSRslct
3250 CMP#&70:BCCeditusr
3260 CMP#&90:BCSeditusr
3270 LDX#(t10-tbs):JSRprtxt
3280 LDA#var2:JSRslct:LDYmemps
3290 STAretmem-&70,Y:JMPrts
3300 :
3310 :
3320 .bysrch LDX#(t11-tbs):JSRprtxt
3330 LDY#0:.lpbysrch
3340 LDX#(srchline DIV256):TYA:CLC
3350 ADC#(srchline MOD256):BCCcclbysrch
3360 INX:.cclbysrch STYvar2
3370 JSRslctntzp:BEQoutbysrch
3380 LDYvar2
3390 INY:CPY#10:BNElpbysrch:INCvar2
3400 .outbysrch LDYvar2:TYA:PHA
3410 JMPbgsrch
3420 :
3430 .stsrch LDX#(t6-tbs):JSRprtxt
3440 .gtstr LDA#10:STAlim:JSRrdline
3450 LDYlgth:.tran LDAinline,Y
3460 STAsrchline,Y:DEY:BPLtran
3470 LDAlgth:BEQgtstr:PHA
3480 :
3490 .bgsrch LDX#(t12-tbs):JSRprtxt
3500 LDA#memps+1:JSRslct
3510 LDA#memps1+1:JSRslct:INCmemps1+1
3520 PLA:STAlgth:JSRonw
3530 LDY#0:.lpsrch
3540 LDAsrchline,Y:JSRprnm
3550 INY:CPYlgth:BNElpsrch
3560 JSRonw:LDA#0:STAmemps
3570 LDA#14:JSRowr
3580 .lpsrch JSRonw:JSRprwrd
3590 .lpsrch1 LDY#&FF
3600 .lpsrch2 INY:CPYlgth:BEQfound
3610 LDAsrchline,Y:CMP(memps),Y
3620 BNEoutsrch:BEQlpsrch2
3630 .found JSRonw:LDA#&81:JSRowr
3640 JSRprwrd
3650 .outsrch INCmemps:BNElpsrch1
3660 LDX#&8F:JSRkeys:BCSensrch
3670 INCmemps+1:LDAmemps+1
3680 CMPmemps1+1:BNElpsrch
3690 .ensrch LDA#15:JSRowr:JMPkeyrts
3700 :
3710 :
3720 .move LDX#(t13-tbs):JSRprtxt
```

```
3730 LDA#memps:JSRslctwrd
3740 LDA#amnt:JSRslctwrd
3750 LDA#memps1:JSRslctwrd
3760 LDAmemps+1:CMPmemps1+1
3770 BCCmoveup:BNEmovedown
3780 LDAmemps:CMPmemps1:BCCmoveup
3790 .movedown
3800 LDY#0:LDXamnt+1:BEQmovedlo
3810 .movedlp
3820 LDA(memps),Y:STA(memps1),Y
3830 INY:BNEmovedlp
3840 INCmemps+1:INCmemps1+1
3850 DEX:BNEmovedlp
3860 .movedlo LDXamnt:BEQenmove
3870 .movedlolp
3880 LDA(memps),Y:STA(memps1),Y
3890 INY:DEX:BNEmovedlolp:BEQenmove
3900 .moveup
3910 LDAamnt:CLC:ADCmemps1:STAmemps1
3920 LDAamnt+1:ADCmemps1+1:STAmemps1+1
3930 LDAamnt:CLC:ADCmemps:STAmemps
3940 LDAamnt+1:ADCmemps+1:STAmemps+1
3950 DECmemps+1:DECmemps1+1
3960 LDY#255:LDXamnt+1:BEQmoveulo
3970 .moveulp
3980 LDA(memps),Y:STA(memps1),Y
3990 DEY:CPY#255:BNEmoveulp
4000 DECmemps+1:DECmemps1+1
4010 DEX:BNEmoveulp
4020 .moveulo LDXamnt:BEQenmove
4030 .moveulolp
4040 LDA(memps),Y:STA(memps1),Y
4050 DEY:DEX:BNEmoveulolp
4060 .enmove JMPrts
4070 :
4080 :
4090 .prompt LDX#(t14-tbs):JSRprtxt
4100 LDAprmvar:BEQturnon
4110 LDA#ASC"F":JSRowr:JSRowr
4120 DECprmvar:BEQenprompt
4130 .turnon LDA#ASC"N":JSRowr
4140 INCprmvar:.enprompt JMPkeyrts
4150 JMPkeyrts
4160 :
4170 :
4180 .jsr LDX#(t15-tbs):JSRprtxt
4190 LDA#memps:JSRslctwrd:JSRonw
4200 LDA#jsra:JSRslct
4210 LDA#jsrx:JSRslct
4220 LDA#jsry:JSRslct
4230 TAY:JSRonw:LDAjsra:LDXjsrx
4240 JSRjsrvec:STAjsra:STXjsrx
```

```
4250 LDX#(t15-tbs):JSRprtxt
4260 JSRprwrd:JSRonw
4270 LDAjsra:JSRprnm:LDAjsrx:JSRprnm
4280 TYA:JSRprnm
4290 LDX#(t7-tbs):JSRprtxt:JSRrd
4300 TSX:INX:INX:TXS:JMPreset
4310 .jsrvec JMP(memps)
4320 :
4330 :
4340 .dis LDX#(t1-tbs):JSRprtxt
4350 LDA#memps:JSRslctwrd:LDA#0:STAlim
4360 .dis1 LDA#0:STAcnt
4370 JSRonw:JSRonw
4380 .bgdis JSRprwrd
4390 JSRspc:LDY#0
4400 LDA(memps),Y:STAopcode:INY
4410 LDA(memps),Y:STAbyte1:INY
4420 LDA(memps),Y:STAbyte2
4430 LDYopcode:LDAmnemvec,Y:TAY
4440 LDX#3:.lpdis
4450 LDAmnemstr,Y:JSRowr
4460 INY:DEX:BNElpdis
4470 JSRspc:LDYopcode:LDAadrmode,Y
4480 CMP#ASC"I":BEQimp
4490 CMP#ASC"#":BEQimm
4500 CMP#ASC"A":BEQacc
4510 CMP#ASC"S":BEQabs
4520 CMP#ASC"Z":BEQzpg
4530 CMP#ASC"X":BEQabx
4540 CMP#ASC"Y":BEQaby
4550 CMP#ASC"x":BEQzpx
4560 CMP#ASC"y":BEQzpy
4570 CMP#ASC"N":BEQind
4580 CMP#ASC"(":BEQiny
4590 CMP#ASC")":BEQinx
4600 :
4610 .imp LDA#1:JMPmtdis
4620 .imm JSRowr
4630 JSRprby1:JMPtwo
4640 .acc JSRowr:LDA#1:JMPmtdis
4650 .abs JSRprby2:LDA#3:JMPmtdis
4660 .zpg JSRprby1:JMPtwo
4670 .abx JSRprby2:JSRx
4680 LDA#3:JMPmtdis
4690 .aby JSRprby2:JSRy
4700 LDA#3:JMPmtdis
4710 .zpx JSRprby1:JSRx:JMPtwo
4720 .zpy JSRprby1:JSRy:JMPtwo
4730 .ind JSRopbr:JSRprby2
4740 JSRclbr:LDA#3:JMPmtdis
4750 .iny JSRopbr:JSRprby1
4760 JSRclbr:JSRy:JMPtwo
```

```
4770 .inx JSRopbr:JSRprby1
4780 JSRx:JSRclbr
4790 :
4800 .two LDA#2
4810 .mtdis STAlgth:CLC:ADCmemps
4820 STAmemps:BCCccldis:INCmemps+1
4830 .ccldis LDA#31:JSRowr
4840 LDA#20:JSRowr
4850 LDA#134:JSRoby:TYA:STAvar2:JSRowr
4860 LDY#0:.lpdis1:LDAopcode,Y:JSRasc
4870 INY:CPYlgth:BNElpdis1
4880 LDA#31:JSRowr:LDA#25:JSRowr
4890 LDAvar2:JSRowr:LDY#0:.lpdis2
4900 LDAopcode,Y:JSRprnm
4910 INY:CPYlgth:BNElpdis2
4920 INCcnt:LDAlim:BEQpage
4930 CMPcnt:BNEcont:RTS
4940 .page LDA#&07:BITcnt:BNEcont
4950 .shift LDX#&FF:JSRkeys:BCScont
4960 LDX#&8F:JSRkeys:BCCshift:JMPrts
4970 .cont JSRonw:JMPbgdis
4980 :
4990 .prby2 LDA#ASC"&":JSRowr
5000 LDAbyte2:JSRprnm:JSRback
5010 BNEmtprbyte
5020 .prby1 LDA#ASC"&":JSRowr
5030 .mtprbyte LDAbyte1:JSRprnm:JMPback
5040 :
5050 .x LDA#ASC",":JSRowr
5060 LDA#ASC"X":JMPowr
5070 .y LDA#ASC",":JSRowr
5080 LDA#ASC"Y":JMPowr
5090 :
5100 .opbr LDA#ASC"(":JMPowr
5110 .clbr LDA#ASC")":JMPowr
5120 :
8000 .t8 EQUD&7F7F7F7F:EQUD&2B17001F
8010 .t9 EQUB&0D:EQUS"70-8F":EQUB&86:EQ
US"?&+"
8020 .t10 EQUS"= &+"
8030 .t11 EQUS"Bytes":EQUW&2B86
8040 .t12 EQUB&0D:EQUS"St Fn":EQUW&2B0D
8050 .t13 EQUS"Strt Amnt Dest":EQUW&2B0
D
8060 .t14 EQUS"Prompt":EQUB&86:EQUS"O+"
8070 .t15 EQUD&0D:EQUS"A  X  Y JSR":EQU
B&86:EQUS"&+"
8080 .mnemstr
8090 EQUS"ADCANDASLBCCBCSBEQBITBMI"
8100 EQUS"BNEBPLBRKBVCBVSCLCCLDCLI"
8110 EQUS"CLVCMPCPXCPYDECDEXDEYEOR"
8120 EQUS"INCINXINYJMPJSRLDALDXLDY"
8130 EQUS"LSRNOPORAPHAPHPPLAPLPROL"
```

```
8140 EQUS"RORRTIRTSSBCSECSEDSEISTA"
8150 EQUS"STXSTYTAXTAYTSXTXATXSTYA"
8160 EQUS"---"
8170 .mnemvec
8180 EQUD&A8A8661E:EQUD&A80666A8
8190 EQUD&A806666C:EQUD&A80666A8
8200 EQUD&A8A8661B:EQUD&A80666A8
8210 EQUD&A8A86627:EQUD&A80666A8
8220 EQUD&A8A80354:EQUD&A8750312
8230 EQUD&A8750372:EQUD&A8750312
8240 EQUD&A8A80315:EQUD&A87503A8
8250 EQUD&A8A80384:EQUD&A87503A8
8260 EQUD&A8A8457B:EQUD&A86045A8
8270 EQUD&A8604569:EQUD&A8604551
8280 EQUD&A8A84521:EQUD&A86045A8
8290 EQUD&A8A8452D:EQUD&A86045A8
8300 EQUD&A8A8007E:EQUD&A87800A8
8310 EQUD&A878006F:EQUD&A8780051
8320 EQUD&A8A80024:EQUD&A87800A8
8330 EQUD&A8A8008A:EQUD&A87800A8
8340 EQUD&A8A88DA8:EQUD&A8908D93
8350 EQUD&A89FA842:EQUD&A8908D93
8360 EQUD&A8A88D09:EQUD&A8908D93
8370 EQUD&A8A28DA5:EQUD&A8A88DA8
8380 EQUD&A85A575D:EQUD&A85A575D
8390 EQUD&A8965799:EQUD&A85A575D
8400 EQUD&A8A8570C:EQUD&A85A575D
8410 EQUD&A89C5730:EQUD&A85A575D
8420 EQUD&A8A83339:EQUD&A83C3339
8430 EQUD&A83F334E:EQUD&A83C3339
8440 EQUD&A8A83318:EQUD&A83C33A8
8450 EQUD&A8A8332A:EQUD&A83C33A8
8460 EQUD&A8A88136:EQUD&A8488136
8470 EQUD&A863814B:EQUD&A8488136
8480 EQUD&A8A8810F:EQUD&A84881A8
8490 EQUD&A8A88187:EQUD&A84881A8
8500 .adrmode
8510 EQUS"I)IIIZZII#AIISSI"
8520 EQUS"Z(IIIxxIIYIIIXXI"
8530 EQUS"S)IIZZZII#AISSSI"
8540 EQUS"Z(IIIxxIIYIIIXXI"
8550 EQUS"I)IIIZZII#AISSSI"
8560 EQUS"Z(IIIxxIIYIIIXXI"
8570 EQUS"I)IIIZZII#AINSSI"
8580 EQUS"Z(IIIxxIIYIIIXXI"
8590 EQUS"I)IIZZZIIIIISSSI"
8600 EQUS"Z(IIxxyIIYIIIXII"
8610 EQUS"#)#IZZZII#IISSSI"
8620 EQUS"Z(IIxxyIIYIIXXYI"
8630 EQUS"#)IIZZZII#IISSSI"
8640 EQUS"Z(IIIxxIIYIIIXXI"
8650 EQUS"#)IIZZZII#IISSSI"
8660 EQUS"Z(IIIxxIIYIIIXXI"
```

Understanding Data Files

This month's First Course is contributed by Paul Pibworth who explains how data is stored in data files by Basic.

I write programs to process data stored in files, and on the whole, the file structure remains the same. But every now and again, I modify the structure, and then sometimes find that the program will no longer read the data from a previous data file. What I needed was a program to read any data file, assuming it was based on strings, integers, and what I call real numbers (numbers like 2.5), which are stored as so-called floating point numbers. The program listed here is the result. However, the added bonus is that it can help you to understand how these things are stored by Basic, and to learn more about floating point numbers.

Before any data file can be used it must first be opened (and if necessary created as a file on disc). This is done with one of the instructions OPENOUT, OPENIN and OPENUP. When a file is opened it is linked to the program via a *channel number*, and all future references to the file use this channel number rather than the file name. Thus:

```
    F=OPENOUT("MyData")
```

would create and open a new data file called *MyData*, and assign its channel number to the variable 'F'.

To write data to a file, or to read data from a file, Basic provides a number of instructions, but the two most often used are PRINT# and INPUT#. Apart from the reference to the channel number, which immediately follows the '#', these instructions are very similar in use to the more normal PRINT (to screen or printer) and INPUT (from the keyboard). Data may be in the form of integer or floating point numbers, or as strings of characters (see below for more detailed examples). This article is concerned with the way in which these three data types are stored in files, particularly floating point numbers.

In addition, there are two other instructions, BPUT# and BGET# which allow a single byte to be written to or read from a file. These instructions are used by the accompanying program to achieve its purpose but we need not consider them further in our discussions here.

The program listed here is designed to help you to understand how Basic stores data in a file. It also serves as a means for examining any data file created using PRINT#. You don't need to understand exactly how the program works to be able to use it. Before you try out the program it helps to create a short data file, such as *TempDat* below, which can be analysed by this program.

This file can be created easily direct from the keyboard, and will also help to explain the use of some of the instructions introduced earlier. Just type:

```
    N%=12345:R1=12.0625:R2=-1.5
    F=OPENOUT("TempDat")
    P.#F,"This is a string."
    P.#F,"Next is an integer."
    P.#F,N%
    P.#F,"Now some real numbers!"
    P.#F,R1
    P.#F,R2
    CLOSE#F
```

and the file will be created. This is just a sample, but one which includes examples of all three data types. Sample numbers are assigned to suitable variables in the first line to ensure that Basic stores these in memory specifically as integer or floating point numbers. If you run the main program, specifying *TempDat* as the file name, it will show you the contents of the file. We will now examine this aspect in more detail.

If you already know how data is arranged in a data file on disc, just skip this bit. All file data is stored as a series of bytes (8 bit numbers in the range 0 to 255 - e.g. numbers like 00110100). If you have created the data file as described above, one way of looking at it is by typing:

```
    *DUMP TempDat
```

The dump will show each byte of the file as a hexadecimal value, with the corresponding characters (if any) displayed to the left.

In this instance the dump will start with a zero byte &00. This usually means that a character string follows. Then comes another number, in hex, which tells you how long the string is. It is &11 (17 in decimal). On the right of your screen, you will see the character string itself, but written backwards. This is the standard format used by Basic for storing character strings in files when using PRINT#.

An integer follows if the leading byte is &40. You can convert this value into decimal by typing:

```
   PRINT &********
```

where the 8 asterisks are the four pairs of hex digits that follow the &40. Integers always use four bytes when stored in data files.

Finally, for a real number, the leading byte is &FF, (but &80 on an Archimedes). The number itself consists of five more bytes, but you will not be able to convert these into a numeric value as simply as you can with an integer. This is because it uses what is known as a floating point number, which makes it rather more complicated to decipher, but more of this later.

The basis of the program is quite simple. The file is read, byte by byte, using BGET#Z% (which reads one byte at a time). Depending on the value of this byte, the program branches to one of three procedures PROCstring, PROCint, or PROCnum.

Basic stores strings on disc with the characters in reverse order, but the INPUT# statement puts everything the correct way round. When a string is found, this is read (see line 1020) and displayed correctly on screen.

Integers are stored as four bytes, but once the program has identified this type of data it can again be read using the INPUT# statement (see line 1090), and then displayed on the screen.

Now we come to the floating point numbers, indicated by the first byte of &FF. Floating point numbers are in two parts. An example is:

6.023 x 10^23

where the first part, 6.023, is called the *mantissa* (*mant%* in the program), the 23, i.e. as in 10 to the power of 23, is the *exponent* (*exp%* in the program). With binary numbers, the 10 in the exponent is replaced by 2. Again, by using INPUT#, Basic does all the work of converting the five bytes used for a floating point number into the correct value (see line 1230). However, the program also contains additional instructions enabling the structure of a binary floating point number to be shown more clearly, as we shall see.

The five bytes comprising a floating point number are again stored back to front, so that the first thing Basic has to do is to reverse them. A decimal point is understood to be between the first byte and the second (reading from the left).

An example of a binary number might be:

1101.1011

The first part (the '1101') means:

1x8 + 1x4 + 0x2 + 1x1 = 13

A binary fraction like the '.1011' means:

1x1/2 + 0x1/4 + 1x1/8 + 1x1/16 = .6895

In binary, 0.1 means 1/2, and 0.01 means 1/4. Thus, 0.11 means 1/2 + 1/4, which we would recognise as 3/4 or 0.75 (decimal). Every binary fraction is thus composed of a mixture of halves, quarters, eighths, sixteens, thirtyseconds, sixty... you must have got the picture. The problem is that you cannot accurately represent numbers like 0.1 (decimal) in binary using such fractions. You just have to get as close as possible.

So we have arrived at numbers (in binary) like 1101.1011, which is 0.11011011 when divided by 2, four times. Thus 0.11011011 x 2^4 will give us back 1101.1011. The computer calls the .11011011 the mantissa, and the 3 the exponent (I have used decimal 3 for convenience). The decimal point is understood to be in front of a "one" (and hence not stored). Furthermore, since the decimal point *must* be followed by a "one", there is no need for this to be stored - it is only put back to show a negative number.

What about numbers like 0.000111. To move the decimal point to the start of the "ones" is to move it the other way. Thus,

0.000111 = 0.111 x 2^-3

It is a positive number, but with a negative exponent. The exponents are added to &80 (so that the exponent is always represented as a positive number).

The program shows the complete binary form of any floating point number, so let us follow a typical example and see how basic deals with this, say:

    00 00 00 08 81

Extract and reverse:

    81 08 00 00 00

The exponent is 81 - 80 = 1

The decimal point comes between byte 1 and byte two:

    01 . 08 00 00 00

The mantissa is bytes 2 to 5:

    .08 00 00 00

Examine the first mantissa byte '08'. The top bit is not set, and it is therefore a positive number. But now set the top bit, i.e. (&80 OR &O8), giving:

    .88 00 00 00

As a 32 bit binary fraction, this becomes:

    0.10001000 00000000 00000000 00000000

The exponent was 1, 2^1 = 2 Therefore x 2 (or move the point once to the right). We now have:

    1.0001000 00000000 00000000 00000000

which is 1 + 1/16 = 1.0625 in decimal.

A negative number works in exactly the same way, except that the first mantissa byte will be greater than 126. It is numbers between 0 and 1 which are perhaps a little harder.

Let us look at such a number, already reversed:

    77 40 00 00 00

Exponent = &77 - &80 = -3, and the mantissa becomes:

    .40 00 00 00

Set the mantissa top bit:

    .C0 00 00 00

and this becomes:

    0.11000000 00000000 00000000 00000000

This time move the point 3 places left:

    0.00011000 00000000 00000000 00000000

This is 1/8 + 1/16 = 3/16 = 0.1875.

This is the principle of floating point numbers. The program shows any real numbers in a file, first as five hexadecimal digits, then in binary format showing how the point is positioned, and finally the resulting decimal number. Of course, Basic's PRINT# and INPUT# instructions do all this for you, but it can be helpful at times to understand in more detail how the data is stored, or to use a program like the one listed here, to examine in more detail the contents of a data file.

If the program helps you as a utility, good, but if it can help you understand floating point numbers, so much the better! Finally, as with a previous article, I found the book *BASIC ROM User Guide* by Mark Plumbley to be most useful.

```
   10 REM Program ReadDat
   20 REM Version B1.1
   30 REM Author  Paul Pibworth
   40 REM BEEBUG  October 1990
   50 REM Program subject to copyright
   60 :
  100 CLS:string$=STRING$(255,"")
  110 DIM num% 5
  120 DIM mant% 32
  130 INPUT"Filename: "F$
  140 CLOSE#0:Z%=OPENUP (F$)
  150 A%=BGET#Z%
  160 REPEAT
  170 B%=A%:A%=BGET#Z%
  180 IF B%=0 AND A%>0 PROCstring(A%):A%
=BGET#Z%
  190 IF B%=&40 PROCint(A%):A%=BGET#Z%
  200 IF (B%=&FF OR B%=&80) PROCnum(A%):
A%=BGET#Z%
  210 UNTIL EOF#Z%
  220 CLOSE#Z%
  230 END
  240 :
 1000 DEF PROCstring(B%)
 1010 PTR#Z%=PTR#Z%-2
 1020 INPUT#Z%,string$
 1030 PRINT'"STRING, of length ";B%;" :
";string$
 1040 ENDPROC
 1050 :
 1060 DEF PROCint(B%)
 1070 PRINT'"INTEGER : ";
 1080 PTR#Z%=PTR#Z%-2
 1090 INPUT#Z%,int%
 1100 PRINT;int%
 1110 ENDPROC
 1120 :
 1130 DEF PROCnum(B%)
 1140 PRINT'"REAL NUMBER : ";
 1150 num%?4=B%
 1160 FOR I%=1 TO 4
 1170 B%=BGET#Z%:num%?(4-I%)=B%
 1180 NEXT
 1190 FOR I%=0 TO 4
 1200 PRINT;~(num%?I%);" ";
 1210 NEXT:PRINT
 1220 PTR#Z%=PTR#Z%-6
 1230 INPUT#Z%,real
 1240 exp%=?num%-&80:mant%?0=ASC"."
 1250 IF num%?1>127 sign$="-" ELSE sign$
=""
 1260 num%?1=num%?1 OR 128
 1270 FOR byte%=1 TO 25 STEP 8
 1280 FOR I%=0 TO 7
 1290 ?(mant%+byte%+I%)=FNbin(?(num%+1+(
byte% DIV8)),I%)
 1300 NEXT I%,byte%
 1310 PROCprintbin
 1320 IF ?num%>0 PRINT" x2^";exp%;" i.e.
 x";2^exp% ELSE PRINT
 1330 IF exp%>=0 PROCmovepointup
 1340 IF exp%<0 AND num%>0 PROCmovepoint
down
 1350 PROCprintbin:PRINT"  = ";
 1360 PRINT real
 1370 ENDPROC
 1380 :
 1390 DEF FNbin(a%,b%)
 1400 b%=2^(7-b%)
 1410 IF (a% AND b%)=b% THEN =1
```

```
 1420 =0
 1430 :
 1440 DEF PROCprintbin
 1450 PRINT"0";
 1460 FOR I%=0 TO 32
 1470 a%=mant%?I%
 1480 IF a%=46 PRINT;"."; ELSE PRINT;a%;
 1490 NEXT
 1500 ENDPROC
 1510 :
 1520 DEF PROCmovepointup
 1530 IF exp%=0 ENDPROC
 1540 FOR pos%=0 TO exp%-1
 1550 a%=pos%?mant%:b%=pos%?(mant%+1)
 1560 pos%?mant%=b%:pos%?(mant%+1)=a%
 1570 NEXT
 1580 ENDPROC
 1590 :
 1600 DEF PROCmovepointdown
 1610 FOR loop%=1 TO ABS(exp%)
 1620 FOR pos%=32 TO 2 STEP -1
 1630 mant%?pos%=mant%?(pos%-1)
 1640 NEXT:mant%?1=0:NEXT
 1650 ENDPROC
```

Conspicuous Consumption (continued from page 33)

```
  110 ONERROR MODE7:PROCerror
  120 PROCload:MODE4:PROCplot
  130 MODE7:PRINTTAB(3,10)CHR$131"Do you
 want to continue?(Y/N)":IF FNkey("YyNn"
) CHAIN"FConB"
  140 CLS:END
  150 :
 1000 DEFPROCload
 1010 PRINTTAB(5,20)CHR$131"Loading entr
y No. "
 1020 V%=OPENUP"Temp"
 1030 INPUT#V%,lastentry%
 1040 FOR entry%=0 TO lastentry%
 1050 PRINTTAB(24,20);entry%
 1060 INPUT#V%,date$(entry%),mile%(entry
%),mpgall%(entry%),mpglast%(entry%)
 1070 NEXT entry%
 1080 CLOSE#V%
 1090 entry%=0:V%=0
 1100 ENDPROC
 1110 :
 1120 DEFPROCplot
 1130 PROCgrid:PROCplotlines
 1140 MOVE0,-140:PRINT;"Do you want a pr
int out?(Y/N)"
 1150 IF NOT FNkey("YyNn") THEN CLS:ENDP
```

```
ROC ELSE CLS
 1160 PROCgrid:PROCplotlines:VDU29,0;0;2
 1170 REM Insert dump call here
 1180 VDU3:MOVE200,24
 1181 PRINT"Press SPACE to continue"
 1182 Z=GET:ENDPROC
 1200 :
 1750 DEF FNkey(K$)
 1760 LOCALK%:REPEAT:K%=INSTR("@"+K$,GET
$) DIV 2:UNTIL K%:K%=K%-2
 1770 =K%
 1780 :
 1790 DEFPROCerror
 1800 IF V% THEN CLOSE#V%
 1810 IF ERR=17 THEN CLS:PRINTTAB(0,10)"
Do you want to continue?(Y/N)":IF FNkey(
"YyNn") THEN CHAIN"FConB" ELSE CLS:END
 1820 REPORT:PRINT" at line: ";ERL:END
 1830 ENDPROC
 1840 :
 1850 REM ( PROCgrid, PROCplotlines
 1860 REM ( and PROCgriddate exactly
 1870 REM ( as Listing 3 [lines 3420
 1871 REM ( to 3970] except that the
 1880 REM ( following lines should be
 1890 REM ( omitted: 3440,3780,3850
```

Practical Assembler: Basic Variable Storage

by Bernard Hill

Last month we started to look at an example of a slightly larger project - to use the Basic's CALL statement to sort an array. The assembler routine has to find out for itself the type of the array and perform the subsequent sort, so that:

```
CALL sort,name(0),n%     or:
CALL sort,names$(0),n%   or:
CALL sort,name%(0),n%
```

would perform the sort where the value of n% indicates the number of items to be sorted. In order to build a bigger program we're going to use the general techniques I dealt with in the July issue (BEEBUG Vol.9 No.3). In particular, we are going to look first at the problem of a comparison of numbers in assembler, and build a routine which will test the result of this. First, however, we need to look more closely at the storage of reals and strings in Basic (note: this month's First Course article similarly looks at the storage of numbers and strings in data files).

REAL NUMBERS

Real numbers are stored in a base 2 mantissa-exponent form. Put simply, that means that a number (say 24.5) is first converted to binary: 24=11000.100 (remember that 0.1 in binary = 1/2), and then stored as a mantissa * a power of 2, 24=0.110001 x 2^5. Within the 5 bytes allocated to a real these are stored as follows:

Byte 1: exponent + 128, in this example:

```
5+128 = 133 = &85
```

Bytes 2-5 contain the 32 bits of the mantissa (1100 0100 0000 0000....). However, since the first bit of the mantissa must always be a 1, it is used instead for the sign of the complete quantity, 1=negative, 0=positive. So +24.5 has a modified mantissa giving:

```
0100 0100 0000 0000 0000 0000 0000 0000
```

which in hex is:

```
&44 00 00 00
```

Thus the complete +24.5 would be:

```
&85 44 00 00 00
```

and -24.5 would be:

```
&85 C4 00 00 00
```

Although at first glance this floating-point format appears to be complicated, it is in fact a standard method of storing floating point numbers across all computers - often with different numbers of bytes for exponent and mantissa depending on the range and accuracy implemented.

You can experiment with actual storage (verifying the values above) by using Program 1 from last month in the same way as that article checked out integer storage.

STRING STORAGE

On the BBC micro the address of the string variable (as given by Program 1 last month) does not contain the string itself but a String Information Block of 4 bytes:

Bytes 0-1:	The address of the start of the string
Byte 2:	The maximum allocated length
Byte 3:	The actual length

Thus arrays of strings, like integers, take 4 bytes per array element. But there is also the string storage itself, which may be anywhere in memory. Unfortunately, on the Beeb there is no automatic deletion of unwanted bits of strings so that the memory used can be freed for further use (*garbage collection*). When a string's length exceeds the allocated length (as in a$=a$+"EXTENSION") a new RAM area is created to contain the new string and the String Information Block is altered to point to it. The old string is just left stranded, and the memory it uses remains unavailable to the program.

THE SORTING PROGRAM

Since this is a Practical Assembler series we are going to look carefully at the steps taken in the creation of this program, which will be completed in the next article. We shall be using 'top-down' and 'bottom-up' techniques, again as mentioned in the July article.

To start with we are going 'bottom-up'. With the complexity of storage formats mentioned last month for integers, and above for reals and strings, it will be apparent that comparing the values of variables (in assembler) is not a simple task. In fact we shall need to consider the three cases separately:

1. INTEGERS

The presence of the sign bit at the top of the last byte (byte 3, counting from 0) complicates matters. We first have to test the sign bits of both integers and consider the four cases separately:

a. Both are positive
b. Only the first is negative (so it is smaller)
c. Only the second is negative (so the first is greater)
d. Both are negative.

Now remembering that we are all the time going to be dealing in pointers where we know the ADDRESS of the variables, we shall have to use indirected addressing, and I shall assume that 'ptr1' and 'ptr2' are zero-page locations containing the addresses of the two integers.

We can obtain the sign bit as follows:

```
LDY #3
LDA (ptr1),Y   \ 3rd byte has sign
ROL A         \ Roll top bit into C flag
BCS .....     \ if C set it was negative
...etc
```

Similar treatment can be provided for *ptr2* and the second variable, but the cases (a) and (d) above will need us to go on to compare bytes 3 down to 0 aborting the comparison when we find a byte differing.

Look carefully at lines 1600 to 1720 in listing 1 which contain the necessary logic for the integer case. The label *less* is the target if the first is smaller than the second, and *greateq* the target for any other possibility.

2. REALS (FLOATING POINT NUMBERS)

Now surprisingly and in spite of their apparently complex storage scheme, once we have done the comparison for integers, reals follow on quite easily. The difference is that the five bytes are numbered 0 to 4 and the sign byte is at the top of byte 1, so must be tested first to give four cases as above. Also, the most significant bytes of a real are in the lower bytes (the integer's are in the higher), and so we must count upwards rather than down. Lines 1420 - 1590 in listing 1 handle reals.

NOTE: See the HINT at the end of this article for clarification of the macros jcc, jne, jcs etc.

3. STRINGS

This is fairly easy, only being complicated by the fact that:

```
LDY #0:LDA (ptr1),Y
```

gives us the first byte of the ADDRESS of the string, so we shall need another zero-page pointer (ptr3) to store it in. Likewise, *ptr4* is obtained from:

```
LDA (ptr2),Y.
```

Also we will start at the first byte of the string (LDY #0:LDA (ptr3),Y) and carry on comparing until the smaller string length has been reached, or the characters are not equal. The string lengths are found by LDY #3:LDA (ptr1),Y and LDA (ptr2),Y so we find the shortest first and store it for later as the stop point of comparisons. Lines 1730 - 1870 of listing 1 contain this code.

Note, that in this string comparison, the strings "ABC" and "ABCD" are considered equal since we stop comparing after 3 characters. I leave it to you to put in the extra few bytes around line 1870 to make the shorter string the 'lesser' of the two.

PUTTING IT TOGETHER

It's no good writing the bottom-up modules unless you can test them. Program 1 provides such an environment whose sole purpose is to test lines 1410 - 1870. Since all this 'skeleton code' is going to be disposed of after testing, I have been lazy: rather than properly print out a string message about which value is greater I have produced two error messages via BRK (see the June issue - Vol.9 No.2) with arbitrary error number 60: I can trap these and re-run to test more numbers (line 170). Of course to test integers and strings you'll have to replace the 'a' and 'b' in lines 140-150 with a%, b% or a$, b$. Note in passing the normal use of the error handler to check for types not handled (static strings - $a and indirected bytes - ?a) as we shan't need these in the sort routine.

Next time we'll be completing the project with 'top-down' methods and incorporating lines 1410-1870 into the final program. If you keep to the line numbering of listing 1 you'll find this easier.

HINTS & TIPS

When developing program code you will of course use situations such as:

```
LDA value:CMP #5:BEQ valueis5
```

where the label *valueis5* occurs later in the program. As your code expands when you develop it, the distance from the BCC instruction to the label often exceeds 128 bytes and during compilation you obtain an "Out of range" error. Naturally you can replace the line above with:

```
   LDA value:CMP #5:BNE over:JMP valueis5
.over ...
```

but this involves another label (.over) and destroys the visibility of the code. Using the macro facility we have developed in previous articles you can invent a JEQ (Jump if EQual) instruction as follows:

```
   LDA value:CMP #5:EQUS FNJEQ(valueis5)
```

by:

```
   30000 DEF FNJEQ(addr)
   30001 [OPT opt
   30002 BNE P%+5 \ branch over next instr
   30003 JMP addr
   30004 ]:=""
```

The same system applies to any of the branch instructions (BPL, BCS etc.) as long as you include the opposite (BMI, BCC etc) in line 30002 above. This month's program uses this technique.

```
   10 REM Ordering test
   20 REM Version B1.0
   30 REM Author  Bernard Hill
   40 REM Beebug  October 1990
   50 REM Program subject to copyright
   60 :
  100 DIM order 256
  110 PROCassemble
  120 ON ERROR GOTO 170
  130 :
  140 INPUT a,b
  150 CALL order,a,b
  160 :
  170 REPORT:IF ERR=60 THEN PRINT:RUN
  180 PRINT" at line ";ERL:END
  190 :
 1000 DEF PROCassemble
 1010 ptr1=&70:ptr2=&72
 1020 ptr3=&74:ptr4=&76
 1030 FOR opt=0 TO 2 STEP 2
 1040 P%=order
 1050 [OPT opt
 1060 LDA &601:STA &70
 1070 LDA &602:STA &71
 1080 LDA &604:STA &72
 1090 LDA &605:STA &73
 1100 LDA &603:CMP #5  \ which type?
 1110 BEQ reals
 1120 CMP #4
 1130 BEQ ints
 1140 CMP #129
 1150 EQUS FNjeq(strings)
 1160 BRK:BRK:EQUS "Type not implemented
yet":BRK
 1170 :
 1410 .reals
 1420 LDY #1
 1430 LDA (ptr1),Y:ROL A:BCS rneg1
 1440 LDA (ptr2),Y:ROL A
 1450 EQUS FNjcs(greateq)
 1460 \ both positive
 1470 DEY
 1480 .loop LDA (ptr1),Y:CMP (ptr2),Y
 1490 EQUS FNjcc(less)
 1500 EQUS FNjne(greateq)
 1510 INY:CPY #4:BMI loop
 1520 EQUS FNjpl(greateq)
 1530 .rneg1 LDA (ptr2),Y
 1540 ROL A:BCC less
 1550 \both negative
 1560 DEY
 1570 .loop LDA (ptr2),Y:CMP (ptr1),Y
 1580 BCC less:BNE greateq
 1590 INY:CPY #4:BMI loop:BPL greateq
 1600 .ints LDY #3
 1610 LDA (ptr1),Y:ROL A:BCS ineg1
 1620 LDA (ptr2),Y:ROL A:BCS greateq
 1630 \ both positive
 1640 .loop LDA (ptr1),Y:CMP (ptr2),Y
 1650 BCC less:BNE greateq
 1660 DEY:BPL loop:BMI greateq
 1670 .ineg1 LDA (ptr2),Y
 1680 ROL A:BCC less
 1690 \both negative
 1700 .loop LDA (ptr2),Y:CMP (ptr1),Y
 1710 BCC greateq:BNE less
 1720 DEY:BPL loop:BMI less
 1730 .strings
 1740 LDY #3 : LDA (ptr1),Y
 1750 STA len \ length of string
 1760 LDA (ptr2),Y:CMP len \ length2
 1770 BCS sover1:STA len \ min length
 1780 .sover1
 1790 \ now get ptrs to strings
 1800 LDY #0:LDA (ptr1),Y:STA ptr3
 1810 INY:LDA (ptr1),Y:STA ptr3+1
 1820 LDA (ptr2),Y:STA ptr4+1
 1830 DEY: LDA (ptr2),Y:STA ptr4
 1840 \ start scanning string
 1850 .loop LDA (ptr3),Y:CMP (ptr4),Y
 1860 BCC less:BNE greateq
 1870 INY:CPY len:BNE loop:BEQ greateq
 1880 .less BRK:EQUB 60:EQUS "<":BRK
 1890 .greateq BRK:EQUB 60:EQUS ">=":BRK
 1900 .len EQUB 0
 1910 ]:NEXT:ENDPROC
 1930 DEFFNjcs(addr):[OPT opt:BCC P%+5
 1940 JMP addr:]:=""
 1950 DEFFNjcc(addr):[OPT opt:BCS P%+5
 1960 JMP addr:]:=""
 1970 DEFFNjeq(addr):[OPT opt:BNE P%+5
 1980 JMP addr:]:=""
 1990 DEFFNjne(addr):[OPT opt:BEQ P%+5
 2000 JMP addr:]:=""
 2010 DEFFNjpl(addr):[OPT opt:BMI P%+5
 2020 JMP addr:]:=""
```

512 Forum

by Robin Burton

This month I'm going to cover one of the several topics which has been on my list of items for the Forum for months, but until now something else has always seemed to crop up instead. At the same time I'll recap on a point I covered in one of the very first issues of Forum, for the benefit of newer readers.

CGA EMULATION

A week or two ago I had a query from a reader which prompted this item. I won't be spending long on it, but it might be interesting for one or two of our newer readers. After all, not all 512 users write programs even in BBC mode.

Essentially my correspondent said that he was delighted with the speed of his 512 (and a friend with an Amstrad PC wasn't quite so pleased), but of course when it came to coloured text displays the Amstrad owner regained his smile. Couldn't something be done to produce colour in 80 column screen modes for the 512, since it's supposed to emulate COLOUR Graphics Adaptor output?

The short answer is no, but the reasons can be presented in two versions. The simple answer is that the 512's 80 column display uses a (BBC) mode 3 screen and that's a two colour display. Fine, but it doesn't explain why (by the way, even if you don't use 'PCSCREEN 7', which is a completely accurate mode 3 screen, the 512 still uses mode 3, but with some direct programming of the BBC's 6845 CRTC to close up the gaps between the lines).

Also as an aside, for those who wonder what a 'single colour' or monochrome display is, strictly this refers to the monitor, not to the computer's display capabilities. After all, a display in the same colour as the background wouldn't be a lot of use, would it?

For the longer version of the answer to this query you need to consider how characters are displayed on the screen by the BBC micro since that is doing the work. If you look at the BBC micro's VDU23 character definitions you can see that each character is made up of 8 bytes in an 8 x 8 bit matrix. If a bit is set to 1 (on) the pixel is displayed in foreground colour, if the bit is off its corresponding pixel is left as background.

In a BBC mode 3 display you'll know that the screen map occupies 16K of RAM. The reason is that if every character takes 8 bytes or 64 bits and there are 25 lines of 80 characters each, it obviously takes 8 x 25 x 80 bytes (= 16,000) to store all the pixels. Each bit of this data can be set on or off (to represent foreground or background) but the problem with displaying more colours is that each pixel must instead be capable of being displayed in any one of the chosen number of colours.

Simple arithmetic therefore shows that to display 80 column text in, say, only four colours, would require that each pixel was represented by one of five values in memory including the background. Even if it was optimised this would take a minimum of three bits (binary values 0 to 4 inclusive) to store and since three bit arithmetic isn't exactly convenient, four bits would be used. This means that the necessary storage for four colours would be 64K bytes. That figure is the entire RAM of the BBC micro including the MOS's memory, and unfortunately shows the absolute impossibility of the idea. If you perhaps thought the 512 or even DOS Plus was the limiting factor, now you know.

There's one final point. Before anyone says "But I have 64K of free sideways RAM", remember that this might not be true for model B and B+ hosts, so Acorn couldn't produce DOS Plus under that assumption. The result is that even if you somehow managed to devise a very clever scheme to intercept screen mapping (?), DOS doesn't pass the colour information across the tube anyway, so it still wouldn't work.

REDIRECTION - INPUT

At last! I've been meaning to talk about this for a long time.

One facility found in the BBC micro which is at first sight missing from DOS is a facility to

*SPOOL (or some similar operation) screen output directly to disc to create a file of the display.

Such a facility would be just as useful in DOS as it is in the BBC micro. You could use it, for example, to create disc files of directory displays, perhaps for reading into a word processor for record keeping purposes. Alternatively you might have text files containing embedded Tab characters and so on, but which you would like to transfer to disc with all the codes expanded to give a formatted file just as it would be if printed.

Well, for those of you who didn't know, it can be done using a standard DOS facility called 'REDIRECTION' and it's very easy to use. First though (as usual) a bit of explanation about how it works.

Those of you who program in DOS will know that everything except the processor and main RAM is, or can be treated as a peripheral, including the screen and the keyboard. This is the philosophy that permits redirection. Of course the various device types which may be attached to a DOS system vary greatly in their control needs and capabilities, but the core of DOS, the BDOS, doesn't need to know about that. All the BDOS is concerned with is memory management and input/output at the most elementary level.

In simple terms, when you enter a command to type a file to the screen the BDOS makes a request through the BIOS (or the XIOS in the case of the 512) to get the required data. The BIOS actually does the disc controller handling, and the disc controller simply passes the raw data to the BDOS. When a sufficient quantity of data has been received (e.g. a 'cluster full') the BDOS then tells the BIOS to output the data to the standard output device, which is usually the screen, but it needn't be, more of which shortly.

What the type command does not specify in this example is the output device, but if you don't specify it, the console output device is used by default. Likewise the default console input device is the keyboard, which is used when it's not specified, but again this need not be so.

I'd guess most of you have at some time used a command in the form 'COPY CON filename' to create a short .BAT file or text file on disc. What this command really translates to within DOS is:

"Read data from the default console input device (i.e. CON, the keyboard) and write the output data to the default console output device (the screen), but also to the named file".

Without sensible device defaults DOS machines would be very difficult to use, so the standard start-up defines the console input and output devices, plus disc as the filing medium, the printer as the hard copy device, communications ports as auxiliary devices and so on. However, all these very different peripherals are just devices and the BDOS is equally happy to 'talk' with one of them as with another. For example, if you enter:

```
    COPY FILE1.TXT FILE1.BAK
```

DOS will assume (unless you tell it otherwise) that the filing system is to be used for this operation and the file is copied from one disc file to another.

Likewise if you simply enter:

```
    DIR
```

DOS assumes that the current directory list is to be output to the screen since no destination was supplied.

However, the way you tell DOS that a different source or destination is to be used is simply by inserting the "<" or ">" symbols respectively at the appropriate point in the command. An example will show this more clearly.

With a (writeable) disc in the current drive try entering:

```
    DIR >CATALOG.TXT
```

and you'll see that the disc starts up in the usual way, but no directory list appears on screen. After a bit more disc activity the drive stops and the normal DOS prompt re-appears. You'll no doubt have guessed where the data has gone to and you're quite right. It has been re-directed into a disc file called 'CATALOG.TXT' (of course, with an example like this the catalogue might be instantly out of date, as you've just added a new file).

After trying that command enter:

```
    TYPE CATALOGUE.TXT
```

which needs no explanation, and the directory list will appear on the default output device, the screen. The console output data from the original command was re-directed into the disc file, which is why it wasn't seen on screen, but the point is you can now edit or manipulate the contents of that file in any way you please.

REDIRECTED OUTPUT

OK, that explains redirected output, but what about input? Simple, using the left chevron, "<", the direction of the operation is reversed. Again an example will best explain it.

Suppose we have a program which, after loading, normally requires one or more lines of input. However, because the program's author was lazy and didn't want to process disc files he didn't provide the option for us to supply a filename to allow automatic reading of input data. The result: we normally have to enter everything manually every time the program runs.

No we don't! There's a much easier way! Let's call the program 'FILTER' for illustration, and let's also suppose that we have created a file containing all the lines we wish to feed to the program called 'COMMANDS.IN'. All we need to do to persuade DOS to supply the text to the program from the file, exactly as if it had been typed live on the keyboard, is to enter:

```
    FILTER <COMMANDS.IN
```

which says to DOS, 'redirect standard input (i.e. the keyboard) to read data from the disc file COMMANDS.IN and pass the data to the program called FILTER.'

INPUT & OUTPUT

Ok, now let's extend our example. Suppose that FILTER reads the input and, after performing its operations, perhaps ensuring that everything is in lower case or that there are no embedded control codes, it displays the data to the standard output device (i.e. the screen). What we want to do though, is to capture a file of the output instead of just getting a temporary display.

No problem! Let's call the new file 'COMMANDS.OUT'. All we need to do is to enter:

```
    FILTER <COMMANDS.IN >COMMANDS.OUT
```

and the input file, COMMANDS.IN, will be read, processed and written to the new output file, COMMANDS.OUT, automatically.

Of course, these are simple examples and this one may look a bit artificial, but even so I'd bet that virtually everyone who didn't know about redirection before can easily think of plenty of occasions when they wish they had known.

OTHER DEVICES

You'll notice that I've been talking about only the standard input and output devices in these examples. That's only because these are the most easily understood for the purpose of explanation. Re-direction can be applied to any device DOS knows about within the limits of common sense (e.g. don't try reading the printer).

Taking our last example again, if instead of creating an output file you wanted hard copy there are several ways you might go about it. You might create the file as above then print it as a separate operation, but if the file is only a means to an end that way is a bit long-winded. In some cases you might be able to use Ctrl-P to enable the printer while displaying to the screen, but you must usually be on the command line to do this. It frequently doesn't work within programs, so you must do the Ctrl-P before you begin, with the result that you also get the commands on your printout, not always an acceptable result.

Using redirection for other than filenames, or instead of the standard input or output devices is easy. The logical printer in the 512, for example, is known as 'PRN:', so to redirect output from the standard output device to PRN: the command would be:

```
    FILTER <COMMANDS.IN >PRN:
```

If output to the serial port were required instead the command would be:

```
    FILTER <COMMANDS.IN >AUX:
```

That's it, again I've run out of space, so I'll have to leave the rest of this subject until next month. In the meantime try a few experiments for yourself. B

Basic Music Composition

Alan Wrigley reviews a new music package from Tobin Music

Product	Basic Music Composition
Supplier	Tobin Music The Malthouse, Knight Street, Sawbridgeworth, Herts CM21 9AX. Tel. (0279) 722318.
Price	£19.95 inc. VAT

Basic Music Composition claims to be an invaluable introduction to learning the fundamentals of composing. It will be of most interest to younger users, though adults may also find it fascinating.

The package consists of a single 40/80 track disc and a slim, though clearly written manual. Also available from Tobin is a *Composition Manual* (price £5.00) which covers the theory of composition in greater detail. The program disc is protected and dire warnings are given about software piracy.

USING THE PACKAGE

On booting the disc, you are first offered the option of using a joystick instead of the keyboard. Since most people will probably use the keyboard, it would have been nice if this could have been built in as a default, instead of having to make the choice each time you boot the disc. The program is driven from a main menu, from which you can choose to create or edit a tune, load or save a tune from or to disc, play the tune currently in memory or print the score on a printer, or select the "instruments" on which it is to be played. An option is also available to allow you to use an extra sound channel which will make the output compatible with Hybrid's Music 5000 Synthesiser Universal (see the review in BEEBUG Vol.9 No.4).

I should make it clear at this stage that the tunes you can create with this package are of a fixed format, in a choice of three keys (C, F or G) and two time signatures (3/4 or 4/4). The software is designed to teach the fundamental principles of composition at a basic level, not to produce a *magnum opus*.

The heart of the program is the main composition screen on which all compositions are made and tunes displayed (see Figure 1). This shows two sets of staves, into which the notes forming the tune are inserted. At the foot of the screen is a further small stave showing the key selected and the notes available in that key, and a small menu which allows you to edit or play the tune, or insert passing notes (more of this later). While you are composing, a small bird dangles a note from its beak to show where you have chosen to place the next note. The bird has other functions in the program, as you will see!

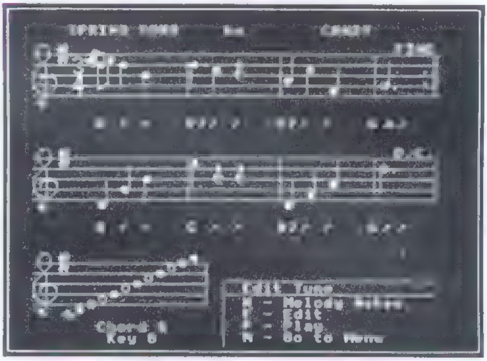


Figure 1. The main composition screen showing a tune ready to be played or edited

COMPOSITION

As explained earlier, all tunes have a fixed format. The first stave is played twice, followed by the second and finally the first again. The melody notes are input by the user, but the harmony is played by the computer in the form of chords in a specific key depending on the bar currently being played. Thus in the key of G, for example, the accompanying chords are G, D7, D7, G for the four bars respectively of the first stave, and G, C, D7, G for the second stave. This means that the melody notes must harmonise with the accompaniment, and thus on each bar you are restricted to only those notes which are applicable. The small stave at the bottom of the screen shows which these are, and any attempt to select a wrong note is disallowed.

The bird can be moved around with the cursor keys, and pressing Return positions a note at that point (provided it is legal). Rests can also be included at any point by pressing the space bar. When the composition is complete, you can edit it at will, or play it in one of a range of tempos from 1 (slow) to 9 (fast). As the tune plays, the bird moves along at the top of the stave to show you which note is being

Continued on page 57

BEEBUG Education

by Mark Sealey

Product	Hands on Spelling, for the BBC B and Master 128
Supplier	ESM, Duke Street, Wisbech, Cambridgeshire PE13 2AE. Tel. (0945) 63441
Price	£27.50 plus VAT, p&p £2.50

This month BEEBUG Education looks at a piece of software designed for use in Primary and perhaps some Special schools, although it could be of some use on Adult Literacy (and similar) courses.

Good new software for the BBC micro is becoming increasingly rare; good software that is really based on wholly sound educational research is usually of special interest in its own right. *Hands on Spelling* falls happily into both categories though there are limitations. What is more, it is definitely not one of those deadly drill and practice programs which rely exclusively on memory and do little to really and effectively support children, who need to see for themselves the whys and wherefores of what they are doing.

BACKGROUND

A bit of background on the rationale behind this package is needed. It has been thought for some time that there is a close and predictable relationship between learning to spell and the act of handwriting. English is composed of common clusters of letters (or more technically graphemes) such as the "or" and "se" in "horse", the "sad" and "dle" in "saddle"). When practising creative handwriting patterns (NOT llllll or efefefec etc!) the child's familiarity with these meaningful clusters of letters is actually reinforced.

It is now widely believed that children learn to read (and write) by recognizing and contextualizing the visual shapes of words and letter clusters as much as by decoding them phonically. So it is that this program presents its material following the patterns of visual structure: "were" with "here" and not with "was", which is found near "has" instead.

Much research has clearly gone into frequency lists, ways of exploiting the children's awareness of alphabetical order, and ways of motivating users by encouraging them to have fun.

Indeed, the clear six page manual (the rest of the documentation is taken up with usable resources) sets out some of this rationale right at the start.

CONCEPT KEYBOARD

The last of the principles outlined in the manual is that the program gives pupils expertise in using IT. One of the most successful peripherals used in education has been the Concept Keyboard. *Hands on Spelling* makes full use of the Concept Keyboard: the resources accompanying the manual consist largely of overlays.

It is recommended that the Concept Keyboard be used in preference to the QWERTY one, though this latter option always exists. Each overlay itself includes a QWERTY style layout of the keys together with Space and Return etc. There is a menu option from the main program which sets these (and other) preferences.

USING HANDS ON SPELLING

On booting, then, you are presented with the top level menu. It is the third option that allows these preferences to be set. The other two (selectable in the usual ways) either load an overlay file or begin the program itself. In each case Escape returns to the main menu, but there appears to be no way of terminating the program altogether.

This may not be as trivial as it sounds and could prove expensive. Children don't always behave as their teachers want, and can grow more and more frustrated by not being able to move on. So great did one girl's annoyance with just this sort of situation once become that she resorted to cutting the disc-drive to computer cable with a pair of scissors in order to end a session!

In most respects *Hands on Spelling* behaves as you would wish, although it is necessary to set the options mentioned above individually for each Concept Keyboard overlay loaded. Otherwise choices are made with the cursor keys and confirmed with Return.

So, you load the overlay file - there are 10, each in A4 or A3 size - and each concentrates on 8 letter groups... "ight", "ound", "wor" etc. This means that pupils can work from a maximum of 80 such groups. It would not be wise, from a teaching point of view, though, to ask children to work through more than one or two at each session.

Nor would many teachers want to abandon the children to explore this central part of the program for themselves - as the manual suggests. In some ways the computer running this package is doing what even a good teacher can rarely have time to do - examine the ways in which letters are grouped in English in detail. Nevertheless, there is much that young children will benefit from if it is pointed out to them by a competent reader as it happens.

What happens next is that the letter group (say "alk") is selected by the child (via either keyboard or a function key) and it is displayed in the centre of the screen in yellow. From the alphabet at the top - with or without sound - descend in turn "t", "ch", "st" and "w" in white to form "talk", "chalk" etc.

It is at this point that the program's lack of interactivity becomes more apparent. Albeit on the educationally soundest of lines, the pupil is asked to reproduce the spelling "t-a-l-k" and the success rate can be monitored (rather grandly called "assessment", another selectable option) and some feedback given.

The strongest part of *Hands on Spelling* is as the letter cluster chosen ("alk") is drawn enlarged in yellow at bottom left, though only once, to reinforce its visual impact and hence "teach" its spelling.

As said earlier, much attention has been paid to spelling families and the organisation and grading of material. The software would be more satisfactory, though, if you could include letter groups of your own - "ion" is missing, for instance, and so is "er".

Finally - another option, this - a poem putting the letter group into context is displayed. Although ways in which the child could have more to DO would have to be thought out very carefully, their lack is felt quite keenly by this point.

To some extent this is compensated for by the excellent and exhaustive set of suggestions in the manual of ways to reinforce and support this learning. A strong point of the product, this.

CONCLUSIONS

Hands on Spelling certainly goes some way to meeting the first criterion for selecting ANY piece of software: in some respects it does what only a computer can do - demonstrates letter groups relatively dynamically.

And there are other definite strengths to this package: it is based on good learning principles. It is easy to use and well-documented; there are plenty of valid follow-up and support activities suggested.

But weaknesses are not insignificant either. Given its sparse scope for interaction between the child and the computer, the protected disc seems a little overpriced at £30+. There are other small points: the poems in the book are printed in non-cursive script after so much emphasis is given in the software to getting the cursive habit.

All in all, if this is an area that interests you and you believe in the underlying philosophy, *Hands on Spelling* is worth a look. However, if any of the drawbacks mentioned here set alarm bells ringing, you may prefer to wait for something more substantial and which will involve the user more. B

Configuring the Master Turbo

Derek Gibbons offers some advice on the use of 6502 second processors.

Master 128 owners who have fitted a Turbo co-processor, or those who have fitted an external 6502 second processor, will have realised that programs (and ROMs) fall into three categories:

1. Those such as Bitstik for which the co-processor is essential. These will obviously fail if the co-processor is not 'ON' and they therefore usually contain a suitable check, and inform the user of this requirement.
2. Those such as System Delta which will not work with the co-processor operating. Again, an appropriate check and comment is often included, but this is by no means universal.
3. Those such as View which will function quite happily in the basic micro, but will benefit from the extra speed and/or memory of the co-processor. These will involve no check at all.

Whether the co-processor is currently configured 'ON' or 'OFF' will obviously depend on the previous application. It is therefore desirable to have a simple, and preferably automatic, method of changing the configuration to suit any new application. Unfortunately, a *CONFIGURE command does not take effect until the next Ctrl/Break which, of course, then halts the flow of any user control routine.

However, the !BOOT file shown as listing 1, which uses ideas published previously in BEEBUG to reduce the amount of unwanted information displayed on-screen, can be used to configure the co-processor 'ON' with the minimum of effort by the user, but will take no such action if the co-processor is already 'ON'.

Either way, the simple demonstration program "TUBE":

```
10 PRINT "Page = &";~PAGE;" in co-processor"
```

is chained to confirm that the required action has taken place, but this program could equally be a full-scale Basic program or a set-up program for another language such as View or ViewSheet.

In use, the !BOOT file first checks the Tube status by means of OSBYTE 234. If 'ON', no further action is taken other than chaining "TUBE" after issuing FX calls via OSCLI to set the Break and Shift/Break actions to normal, and to close the !BOOT file so that when "TUBE" has run, control is not handed back to the rest of the !BOOT file. If the Tube status is 'OFF', *CONFIGURE TUBE is issued, the Break and Shift/Break actions are reversed with *FX255, and Break is then simulated with CALL!-4, but is interpreted as a Shift/Break causing the !BOOT file to be re-entered.

Strictly speaking, as mentioned earlier, Ctrl/Break should be used to implement a configure command, and is indeed essential in many cases, but the CALL!-4 appears to be adequate here. This call could be preceded with *FX151,78,127 to give what purports to be a simulated power reset and, although this would give an undoubtedly stronger Break action, it unfortunately also destroys the setting of FX225 before it can be utilised.

The co-processor can equally be turned 'OFF' by a slightly modified !BOOT file, as shown in listing 2.

Listing 1: !BOOT file for co-processor 'ON'

```
*BASIC
CO.0
VDU11,11,11,11,11:PRINT SPC7’SPC5
A%=234:X%=0:Y%=255:t%=(USR(&FFF4) AND
   &FF00) DIV 256
IFt%=255 CO.7:OSCLI"FX255,8,247":
   OSCLI"FX119":CHAIN"TUBE"
*CO.TUBE
*FX255,0,247
CALL !-4
```

Listing 2: !BOOT file for co-processor 'OFF'

```
*BASIC
CO.0
VDU11,11,11,11,11:PRINT SPC7’SPC5
A%=234:X%=0:Y%=255:t%=(USR(&FFF4) AND
  &FF00) DIV 256
IFt%<>255 CO.7:OSCLI"FX255,8,247":
  OSCLI"FX119":CHAIN"HOST"
*CO.NOTUBE
*FX255,0,247
*GOIO E364
```

Here, the status check after OSBYTE 234 has been reversed; the demonstration program "HOST":

```
  10 PRINT "Page = &";~PAGE;" in Host Proc
essor"
```

is chained; and *GOIO E364 is used instead of CALL !-4 to ensure that the appropriate MOS routine is called in the Host processor. Note that the contents of !-4 are &E364 on a Master 128, rather than the more familiar &D9CD of the Model B.

There is, however, a snag when using the DFS! These !BOOT files *MUST* then be called by Shift/Break and not by *EXEC !BOOT or *!BOOT, thus precluding their use on any drive other than drive 0, because the simulated Shift/Break is still a Shift/Break and so attempts to use a !BOOT file on drive 0 regardless. This restriction does not apply to ADFS where Shift/Break will still work on drive 1, if it is the current drive.

However, even with the ADFS, or with DFS using drive 0, if Shift/Break is not used, the *first* Host-to-Tube change after a genuine Break will still work; all Tube-to-Host changes will apparently work; but if a Host-to-Tube change is attempted after a Tube-to-Host change without an intervening genuine Break, the computer will hang until Break is actually pressed, although the changeover will then be found to have been implemented. Using the !BOOT files with Shift/Break provides the necessary Break action.

This problem appears to be due to the fact that the use of *GOIO E364 to simulate a Break (even if preceded by *FX151,78,127) does not perform *ALL* the actions which even a simple Break causes, especially as far as the Tube is concerned. Perhaps someone knows a way around this problem, using a different FX command, or a different MOS entry point, to fully reset the micro when moving from co-processor 'ON' to co-processor 'OFF'.

NOTE: With the alternative Master System ROM fitted, the *GOIO E364 command needs to be changed to *GOIO E374.

Basic Music Composition (continued from page 53)

played. You can then, of course, edit the tune further until it sounds just right when played.

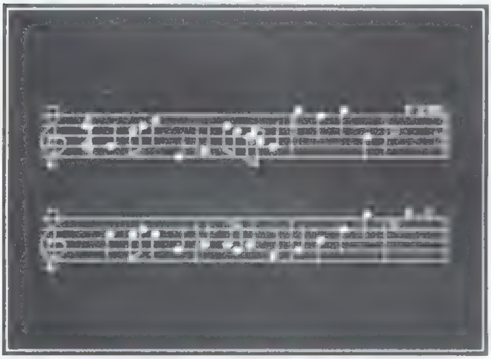


Figure 2. Examples of passing notes (upper stave) and auxiliary passing notes (lower stave)

PASSING NOTES

To increase the versatility of the composition process (and the listenability of your tunes!) it is possible to place passing notes at any point in the tune. A passing note is a note of half the normal length positioned between two notes which are on adjacent lines or adjacent spaces on the stave; when inserted, the length of the previous note is also automatically halved and the two joined together. The program also accepts auxiliary passing notes, which are positioned between two identical notes. Examples of passing notes are shown in Figure 2.

Tunes can be filed on disc, and previously saved tunes can be loaded into memory and then played or edited. The program disc contains a number of specially-composed tunes. You can also choose a demonstration mode, in which the computer will generate a tune at random. Sometimes this turns out to be a quite acceptable melody, but it is very much a matter of pot luck!

Having composed your tune, you can print out the score on an Epson-compatible printer. This is a useful facility and I can imagine many children rushing home from school clutching a printed copy of their latest work to show everyone.

CONCLUSIONS

Basic Music Composition is a well thought-out piece of software which should prove very helpful both in the classroom and at home. Bear in mind, though, that it is designed as a learning aid and not a complete music system, and you should not expect to produce musical masterpieces with it.

Personal Ads

BEEBUG members may advertise unwanted computer hardware and software through personal ads (including 'wants') in BEEBUG. These are completely free of charge but please keep your ad as short as possible. Although we will try to include all ads received, we reserve the right to edit or reject any if necessary. Any ads which cannot be accommodated in one issue will be held over to the next, so please advise us if you do not wish us to do this. We will accept adverts for software, but prospective purchasers should ensure that they always receive original copies including documentation to avoid any abuse of this facility.

We also accept members' Business Ads at the rate of 30p per word (inclusive of VAT) and these will be featured separately. Please send us all ads (personal and business) to MEMBERS' ADS, BEEBUG, 117 Hatfield Road, St. Albans, Herts AL1 4JS. The normal copy date for receipt of all ads will be the 15th of each month.

M128, Turbo co-processor, two CS400S Cumana disc drives, GIS teletext adaptor, AMX mouse, SuperArt, Pagemaker, Viewstore, Interword, Wordwise plus, MicroProlog, Logotron Logo and other firmware. BEEBUG bound vols. 2-8, ref. manuals 1&2 with many other books and discs £500 o.n.o. Tel. (0272) 771733.

BEEBUG Printwise disc and manual, Dabs Hyperdriver ROM and manual, Merlin Database ROM disc and manual, Micro Aid Cashbook disc and manual, Acornsoft Viewsheet ROM and manual, no reasonable offer refused. Tel. (0705) 371018.

BBC Master 512 with 40Mb Viglen hard disc, dual 40/80T DS/DD Technomatic disc drives, Microvitec 653 colour monitor, manuals for Master, DOS+ manuals and books. Software; GEM suite (3 discs), View, Viewsheet, Edit (ROMs), mouse, a few games and discs £740. Tel. (0635) 578925.

Spellmaster £25, Wordwise Plus £20, ADT £15, Artroom, Ace, Digimouse, Chauffeur £30, Speech! £5, Genie Junior £10, Printwise II £10, Master ROM cartridge (ACP) 2 £6 each, Books; Assembly Language Programming (Birnbaum) £6, Programming the 6502 (Zaks) £6, Creative Assembler (Griffiths) £3, Master Sideways RAm User Guide (Smith) £6, all with original documentation etc. prices include p&p. Tel. (0302) 744005.

BBC A3000 computer, colour monitor + lead, printer lead, serial chip, mouse mat, 2 games, demos. The computer system is just over one year old and in very good condition. Asking price is only £770. Willing to throw in an old 9 pin working Star SG-10 printer, if whole set-up is collected within a few days of display of this ad! Tel. John: Welwyn Garden (0707) 323032 eves.

WANTED: Vol. 1 of BEEBUG magazine complete or Nos. 1-6 and 8. Tel. (0243) 862842.

Disabled Master user requires to purchase one or two Peartree MR8000 battery backed RAM cartridges. Due to speech being part of problem please write rather than phone. Mr Skelton, Queens Pit Bungalow, Pit Lane, St Columb, Cornwall,TR9 6LG.

512 co-processor in Acorn case with PSU M-TEC basic "DOS plus reference guide" (Glentop). Norton "Programmers guide to the IBM PC" mouse and GEM software, offers around £175. Tel. (0705) 371018.

Archimedes 410/1, 4Mb RAM, 53Mb hard disc, paper white monitor (only 2 months old), Eizo 9060SZ multisync monitor, Aleph One 30 MHz ARM3 upgrade, Epson LQ2500 printer, Autosketch V2, 1st Word Plus V2, Logistix, Impression DTP, PC Emulator V2, Artisan, absolutely loads of games & utilities, 2 lockable boxes full of blank discs, every issue of Archive & RISC User. All boxed as new for only £2400 (cost over £4600) Tel. 081-997 8484 extn 258.

BBC B issue 7, with ATPL Sideways ROM/RAM board issue 4, (takes 12 extra ROMs) including Wordwise, Printmaster, 2 extra RAM chips and some odd EPROMs, Philips green screen, double plinth and twin double sided 80T disc drives. Offers invited around £395 for the complete system. Also available Raven 20 shadow RAM board, wide carriage Epson serial printer type MX80 FT, 5" disc library boxes (new), offers? Tel. 071-253 4399 extn 3275 or (0487) 814227 eves.

Microvitec 1431 monitor, Torch ZDP dual 40/80 drives with Z80 board etc., various ROM & RAM boards, 5.25" discs, various ROMs. For list phone (0873) 3429.

Overview package, suitable for use with M128, includes Viewspell and Viewstore in a ROM cartridge, with additional Overview facilities, Viewindex, Viewplot, Printer Driver Generator and all supporting programs on an 80T double sided 5.25" disc. Also includes manuals for all the View range ie. View and Viewsheet £45 o.n.o. Tel. (0924) 826483.

BBC B 1.2 plus Cumana mains powered 3.5" 40T disc drive and Shadow RAM, plus CP80 printer in virtually new condition £250 o.n.o. Tel. 071-736 5429.

BBC B 1.2 with shadow RAM, ROM board with Toolkit, Prolog, Gdump, ROMit, Interword, 16K battery backed RAM, Music 500, Shinwa CP80 printer, many games, software, books, BEEBUG magazines & tapes, joysticks. All for £250 (or will split). Tel. (0535) 605793.

Archimedes 440 colour, RISC OS, 1st Word Plus release 2, PC Emulator, mouse mat, Atelier art package, 5.25" disc drive interface, all manuals, boxed as new (less than a month old - 11 months guarantee!) £2300 o.n.o. Tel. (0909) 562399.

Printwise II (5" disc), Dumpout III, Dumpmaster, WYSIWYG Plus, all at £10 each, PCB (2 chip version) £30, Care quad cartridge for Master 128 £10. Tel. (0428) 713326 eves.

BBC B issue 4 with Acorn DFS, 40/80 DS/DD disc drive, Morley Teletext with ATS, Solidisk 2Mb 128 and DFS, Computer Concepts MEGA 3 ROM plus misc. software, books and magazines. All in superb condition. Unbeatable value at £385, could sell separately. Send an sae for full list. Tel. (0628) 825202.

POSTBAG

POSTBAG

MORE VIEWS ON WORD PROCESSORS

I was especially interested to see the recent comparisons of word processors, as I had just been doing an in-depth look at both InterWord and Wordpower. Admittedly, your articles are intended to be a brief resume of each, but in looking at InterWord there are two strong points one for and one against, that I was surprised were not mentioned.

A very powerful feature is the ability to format the page in up to four columns, newspaper style. A strong disadvantage is that there are only four built-in printer codes; after that you have to delve into your printer handbook and enter codes yourself, which is distinctly unfriendly.

Likewise, in Wordpower, the opposite is true. You can easily vary the column width, but can have only one column. But there are 26 single letter codes loaded with the word processor, which can be tailor-made for each user or printer. So I use 'B' for bold, 'C' for condensed, 'D' for double height, and so on.

Many users will prefer the pull-down menus of InterWord, and not bother with using codes other than for Italic or underline, but I certainly prefer the wide variety of print codes easily available in Wordpower.

David Williams

Having just read the survey of Wordwise, View and InterWord, it struck me how muted the praise for the latter was. In my experience it is so vastly superior to all the others for the Beeb that I cannot understand anyone fiddling about with Wordwise or View when the job can be done so much more easily with InterWord, unless perhaps you are a computer buff.

If, like me, you are someone who simply wants a computer to make some things easier and quicker use InterWord. Let me give an example. I have a daisy wheel printer, and with some daisy wheels I need to give it a special code to make it print '£'. Using InterWord that code is entered in the control codes menu, and that is the end of it. With View, I have spent hours trying to achieve the same effect. I'm not saying it can't be done - just that for those uninterested in programming it is a hassle.

J.A.Brook

Each person has their own particular requirements when it comes to choosing a word processor (like anything else). You need to decide what features or facilities are most important to your own work, but even then there are other less tangible factors at work. The answer is often to stay with the product you feel most comfortable with, but not to be afraid of trying a different 'brand' if you can't get on with the one you already have.

MORE TALES OF THE TRAVELLING SALESMAN

Many thanks for publishing my letter requesting help on the Travelling Salesman Problem (Postbag, BEEBUG Vol.9 No.3). I'm pleased to say that I have had a reply which gives an acceptable solution to the problem, and it fits the Beeb (see also BEEBUG Workshop in BEEBUG Vol.9 No.4).

Now for another question. Is there a method of cutting down memory requirements when using less than half a square array? I am thinking of the example of the mileage table for distances between towns given in motoring books and the like. My need is for 30 x 30, which is wasteful when under half is necessary.

John Holt

*This is an interesting challenge which I was unable to resist. First of all, assuming that all data is in integer format, it can be stored and accessed using the '!' indirection operator. Assuming that the table consists of **n** towns, then it can be shown that the distance from town 'a' to town 'b' (assuming that for all a,b b>a) is at:*

```
!(start+4((2n-a)(a-1)/2+(b-a-1)))
```

*where **start** is the address of a reserved area of memory, i.e.:*

```
DIM start 2*(n-1)(n-2))
```

The working out of this is something else! Any other thoughts on this subject would be most welcome.

Master turbo co-processor, a 65C102 co-processor fitted complete in an Acron Universal second processor unit. As new £110 o.n.o. Tel. (0793) 770886 or after 6pm (0793) 771659.

M128 with 80T 5.25" DSDD drive & PSU, Voyger 7 Auto Answer modem and Epson RX80 printer, all manuals included plus View Printer Driver Generator software and View manual, a complete word processing and comms setup for £550 o.n.o. Tel. (0226) 340421.

Unused, Quest mouse MkII, The complete Mouse User Guide and the examples disc from the Complete Mouse User Guide all for £30. Tel. 021-449 7211.

Archimedes colour monitor, almost new £100. Tel. (0428) 713326 eves.

Archimedes 440/1 60Mb hard disc, colour monitor, additional external 5.25" disc drive and modem, all manuals, a wide range of software for Arc, including PC Emulator and several good MSDOS packages. About 1 year old and has had little use, offers in the region of £2200 for the package which could include a Tandy daisywheel printer if required. Tel. (0223) 872178.

ATPL sidewise ROM board (no fitting instructions) £10, Acorn 0.9 DFS kit (8271 FDC) £20, BASIC II ROM £8, OS1.2 ROM £4, Speech processing ROMs (TMS 6100, TMS 5220) £10 the pair, model B power supply £15. Other spares available. Tel. (0244) 344695.

Viglen PC style case for model B with dual sw mode p.s. & 12v fan, £40 o.n.o. Tel. (0992) 583228.

Interword ROM (as new), Spellcheck ROM (as new), Command ROM (as new), Dumpmaster ROM (not used), Printwise disc (not used), Masterfile II disc (not used), all with manuals etc. plus misc. BBC books, 7 games, joysticks, some unused 5.25" discs. The lot sold together £125. Cumana CS100 disc drive £30, Acorn tape recorder as new £25. Tel. (0753) 868038.

Music software, complete music performing/printing package for the BBC B/Master. This hardly used package includes EMR Miditrack Editor, Performer, Scorewriter, Utilities, Midi Interface and all manuals. May split, offers around £230.00. Tel. (0705) 371018.

Model B issue 7 £170, Cumana dual sided double drive 40/80T, PSU £120, Epson LX80 printer £120, Solidisk 8271/1770 DFS £35, Solidisk 4Mb board £80 o.n.o. Tel. (0634) 241237 after 5pm.

M128 turbo, Viglen PC case, twin 40/80 DS drives, RGB monitor, MOS+ ROM, Vine ROM board 3&4, Master Replay ROM, Soundmaster volume control £400. Turbo co-processor £80, Internal modem (BEEBUG) £75, Interword £30, Spellmaster £35, ADI £15, Wordwise Plus £25, various games discs £5 each, Master Reference manuals £8 the pair. Tel. (0780) 53484 eves.

WANTED: Hybrid Music 5000 & 4000 (+ all software and guides). Either will be bought at a reasonable price. Tel. (0794) 68373.

Master 512/1024 (Solidisk PC+) with dual 80T DS DD, 2 joysticks, View etc., cartridge with Printmaster ROM, Dabs M512 Guide, Master Ref. manuals, BEEBUG magazines vol. 1 no. 9 to present, TCS 512 mouse driver, mosaic twin advanced pc Lotus compatible spreadsheet package, considerable amount of BBC & PC software £500. Acorn 30Mb Winchester drive for Master £250, also BBC teletext adaptor & TFS, £30 o.n.o. Tel. (0905) 67488 eves.

8087-2 co-processor, new unused £45, Genius mouse £20, Magic Modem with command ROM £50, BBC B computer with shadow RAM BASIC II, EPROM board and sideways RAM, EPROM ZIF socket, one owner since new £150, Epson FX80+ printer with serial port and NLQ, as new, little used £120, Monochrome, green monitor, long persistence, composite and video + sync £30, Taxan supervision III monitor, as new £150, BEEBUG cassette tapes, vols 1-10 to 8-7 £50, Challenger 3, 80T floppy disc drive with 512k RAM disc, plus advanced disc investigator £120, Morley teletext adaptor with ATS ROM £55, Watford NLQ ROM for Epson £10, Wordwise plus £20, BEEBUG 'C' Language £25, Interbase £35, Interword £30, Intersheet £25, Interchart £16, Spellmaster £35, BROM plus £15, Pen Friend II £15, Clares fontwise plus and editor £15, Clares Grafdisk £5, *GDUMP 2.01 ROM £5, Gemini Database + Beebcalc spreadsheet + Beebplot £7, Psion Vufile + Vucalc £5, Design 7 screen designer £5, Toshiba P351SX dot matrix printer, new & unused £400. Tel. (0276) 35228.

BBC B DFS 1.44 with 32k shadow RAM card, ZIF socket, w/processor, s/sheet, & d/base, Hantarex (green) monitor, Cumana single d/d 40T, Epson RX80 printer £520 o.n.o. Tel. (09274) 24076.

BBC B issue 7 with 32k sideways RAM, Challenger 3 disc drive, B&W monitor, joysticks books & discs £280 o.n.o. Tel. (0959) 71133 eves.

M128 twin 40/80T drives, two switchable ROM cartridges, Master ROM, Teletext adaptor, Wapping editor & mouse, DTP package, Epson compatible (FX80) Datac dot matrix printer, Teletext/mouse splitter box, 8 vols of BEEBUG in binders, Acorn User mags, selection of books on BASIC and computing, software, disc box and discs, computer; disc drives & teletext adaptor mounted in wood cabinet on top of which a monitor will stand. Ideal package for a newcomer to computing. Complete package £575. Tel. 081-304 1405.

BBC issue 7, Acorn DFS, Solidisk dual floppy disc controller (8271/1772) inc. ADFS+DDFS, Viglen 40/80T DSDD, Cumana 40T SSSD (PSU), HCR electronics 16k ROM/RAM expansion board, AMX MkII mouse, Toolkit+, Interword, all manuals + boxes BEEBUG magazines vol.2-8 to vol.9-4 (5.25" disc vol.7-8 to vol.9-4) £600 o.n.o. plus many selected games discs from £7, tape from £1 all original. Tel. (0392) 423596.

Apollo modem & software £40, EPROM programmer (includes some blank EPROMs)k £25, Commstar-1 £15, Printmaster £20, Stop Press, SuperArt, mouse and software to match (originally cost £150) £90, Extra-Extra Utilities £15, Acornsoft LOGO brand new (normally £70) £35, CYB Mailing list £5, Mail Merge £5, 3.5" Elite (Master) £5, 3.5" Music System £10, Master reference Guide Part 1 £5, all software is original and boxed as new, all prices include postage etc. Tel. (092575) 5139 after 6pm except at w/ends.

Book WANTED: Wordwise Plus user guide by Bruce Smith (Collins). Tel. (0935) 812007.

Software & books for BBC B/Master for sale, as new, mostly at quarter of original price, list available, Micro User magazines, vols 1-4 complete in binders £25, A&B computing 1983-86, 32 issues + 3 supplements in binders £20, either set without binders at £8 less. Collect London or St Albans or post extra please. Tel. (0727) 861835.

Mono monitor, Taxan amber screen 12" composite video in £25 o.n.o. Tel. (0582) 581051.

We have a small quantity of BEEBUG Members Pack discs (40 track only) at £2.50 each inc. p&p.

First come, first served!

If you require details, please contact Alan Wrigley at BEEBUG.

PRINTING A '£' SIGN

D.M.Bruce

Printing a '£' sign from Basic (and from other applications) poses difficulties. The routine listed here solves the problem by intercepting the INSV vector and if an insertion is for the print buffer (3) and the character is '£' (ASCII 96) then a string of codes is sent to the printer to change to English characters, send ASCII 35 (which generates '£'), and return to American characters (the default). The program should be saved in the library directory and then run to assemble the necessary code which will have the name *pr£*. At any time the routine can be established by typing:

```
*pr£
```

after which the '£' sign will be printed correctly, until INSV is reset by pressing Break (or Ctrl-Break). The program uses memory from &9A0 to &9FD.

The routine was developed for a Citizen 120D printer, but may well work with other printers (or can be adapted to do so - the relevant codes are contained in lines 230 to 290).

```
 10 REM Program to print '£'
 20 REM Author D.M.Bruce
 30 DIM code% 120:INSV=&22A:start%=&9A0
 40 FOR Z%=4 TO 7 STEP 3
 50 O%=code%:P%=start%
 60 [OPT Z%
 70 .init
 80 LDX #ins AND &FF:CMP INSV:BEQ exit
 90 LDA INSV:STA defV
100 LDA INSV+1:STA defV+1
110 LDY #ins DIV &100
120 SEI:STX INSV:STY INSV+1:CLI
130 .exit
140 RTS
150 .defV
160 EQUW 0
170 .not£
180 PLP:PLA:JMP (defV)
190 .ins
200 PHA:PHP
210 CPX#3:BNE not£
220 CMP#96:BNE not£
230 LDY #27:JSR codein
240 LDY #82:JSR codein
250 LDY #3: JSR codein
260 LDY #35:JSR codein
270 LDY #27:JSR codein
280 LDY #82:JSR codein
290 PLP:PLA:LDA #0
300 JMP (defV)
310 .codein
320 LDA#1:JSR &FFEE
330 TYA:JSR &FFEE:RTS
340 ]
350 NEXT
360 OSCLI("SAVE pr£ "+STR$~code%+" "+
STR$~O%+" "+STR$~start%+" "+STR$~start%
370 END
```

SIMPLE SPRITES

Al Harwood

Fast mode 5 multi-coloured sprites can be added to your programs easily with the use of this routine. Once run, you enter the pixel colours using keys 0 to 3. An enlarged version of the sprite will appear on the screen - I recommend that you draw the sprite on paper first. Once complete, a line of data will appear, which can be added to your own programs, though the line number may need to be changed (see line 500 below).

To display the sprite, first RESTORE to the relevant DATA line, and then call PROCsprite(X,Y) to place the sprite at screen co-ordinates (X,Y), e.g.:

```
RESTORE 500:PROCsprite(3,2)
```

```
 100 MODE 5:DIM P(7,7)
 110 FOR Y=0 TO 7:FOR X=0 TO 7
 120 VDU31,X+5,Y+5:A=GET-48:VDU17,A+128,
32:P(X,Y)=A:NEXT
 130 COLOUR 128:PRINT''"1000D."
 140 FOR A=0 TO 4 STEP 4
 150 FOR B=0 TO 7 STEP 4
 160 T$="":FOR C=0 TO 3
 170 D$=STR$~((P(A+3,B+C)AND1)+
((P(A+3),B+C)AND2)*8)+((P(A+2),B+C)AND1)*2)+
((P(A+2,B+C)AND2)*16+((P(A+1,B+C)AND1)*4+
((P(A+1,B+C)AND2)*32)+((P(A,B+C)AND1)*8)+
((P(A,B+C)AND2)*64))
 180 D$=STRING$(2-LEND$,"O")+D$:T$=D$+T$
 190 NEXT:PRINT"&"T$",";
 200 NEXT,:PRINT CHR$127'
 210 END
 220:
 230 DEF PROCsprite(X%,Y%)
 240 A%=HIMEM+X%*16+Y%*320
 250 READ !A%,A%!4,A%!8,A%!12
 260 ENDPROC
 270:
 500 DATA &03113130,&22330505,&0C88C8C0,&44C
COAOA
```

BEEBUG MEMBERSHIP

Send applications for membership renewals, membership queries and orders for back issues to the address below. All membership fees, including overseas, should be in pounds sterling drawn (for cheques) on a UK bank. Members may also subscribe to RISC User at a special reduced rate.

BEEBUG SUBSCRIPTION RATES		BEEBUG & RISC USER
£16.90	1 year (10 issues) UK, BFPO, Ch.I	£25.00
£24.00	Rest of Europe & Eire	£36.00
£29.00	Middle East	£43.00
£31.00	Americas & Africa	£46.00
£34.00	Elsewhere	£51.00

BACK ISSUE PRICES (per issue)

Volume	Magazine	5"Disc	3.5"Disc
5	£1.20	£4.50	£4.50
6	£1.30	£4.75	£4.75
7	£1.30	£4.75	£4.75
8	£1.60	£4.75	£4.75
9	£1.60	£4.75	£4.75

All overseas items are sent airmail. We will accept official UK orders for subscriptions and back issues, but please note that there will be a £1 handling charge for orders under £10 which require an invoice. Note that there is no VAT in magazines.

POST AND PACKING

Please add the cost of p&p when ordering individual items. See table opposite.

Destination	First Item	Second Item
UK, BFPO + Ch.I	60p	30p
Europe + Eire	£1	50p
Elsewhere	£2	£1

BEEBUG
117 Hatfield Road, St.Albans, Herts AL1 4JS
Tel. St.Albans (0727) 40303, FAX: (0727) 860263
Manned Mon-Fri 9am-5pm
(24hr Answerphone for Connect/Access/Visa orders and subscriptions)

BEEBUG MAGAZINE is produced by BEEBUG Ltd.

Editor: Mike Williams
Assistant Editor: Kristina Lucas
Technical Editor: Alan Wrigley
Technical Assistant: Glynn Clements
Production Assistant: Sheila Stoneman
Advertising: Sarah Shrive
Managing Editor: Sheridan Williams

CONTRIBUTING TO BEEBUG PROGRAMS AND ARTICLES

We are always seeking good quality articles and programs for publication in BEEBUG. All contributions used are paid for at up to £50 per page, but please give us warning of anything substantial that you intend to write. A leaflet 'Notes to Contributors' is available on receipt of an A5 (or larger) SAE.

Please submit your contributions on disc or cassette in machine readable form, using "View", "Wordwise" or other means, but please ensure an adequate written description is also included. If you use cassette, please include a backup copy at 300 baud.

In all communication, please quote your membership number.

BEEBUG Ltd (c) 1990
ISSN - 0263 - 7561
Printed by Arion Printers (0923) 268328

Magazine Disc

October 1990 DISC CONTENTS

SPINNING TEXT - a fascinating visual display, which draws and spins text in 3D - a single word or a short message. Two demonstration files are also provided.

ADFS DISC SECTOR EDITOR - a useful program for repairing corrupted or unreadable discs, which allows you to display and edit individual disc sectors on-screen and save the edited sector back to disc.

PINPOINTING A DATE - a program demonstrating a routine for calculating a future calendar date.

CONSPICUOUS CONSUMPTION (Part 2) - Two programs, for the model B and the Master, to provide plotting facilities for last month's fuel consumption monitor.

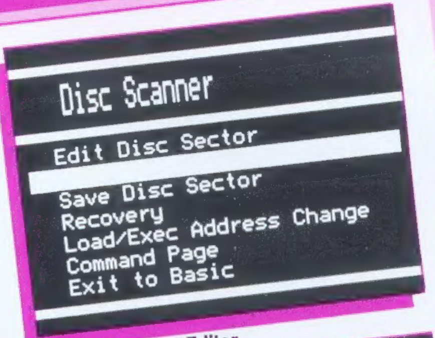
BEEBUG WORKSHOP: Searching (Part 2) - two programs which use the 'Queens problem' and the 'Knight's tour' on the chessboard to demonstrate the use of a backtracking algorithm for simplifying the searching process.

MONIX: A MACHINE CODE MONITOR (Part 2) - the complete program, including last month's utilities and 7 new routines such as memory editor, disassembler, mover, search routine etc.

FIRST COURSE: Understanding Data Files - a program demonstrating the storage of floating point numbers in files.

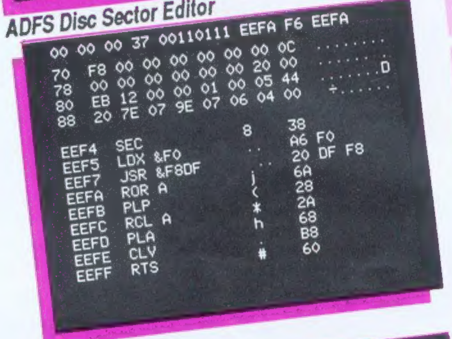
PRACTICAL ASSEMBLER (Part 4): Basic Variable Storage - a demonstration program from this month's article which looks at the storage of numbers and strings in data files.

MAGSCAN DATA - bibliography for this issue (Vol.9 No.5).



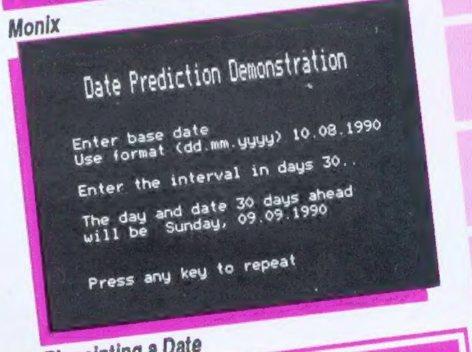
```
Disc Scanner

Edit Disc Sector
Save Disc Sector
Recovery
Load/Exec Address Change
Command Page
Exit to Basic
```

ADFS Disc Sector Editor



```
00 00 00 37 00110111 EEFA F6 EEFA
70  F8 00 00 00 00 00 20 00   ........
78  00 00 00 00 01 00 05 44   .......D
80  EB 12 00 00 01 00 05 44   ÷.......
88  20 7E 07 9E 07 06 04 00

                        8    38
EEF4  SEC                    A6 F0
EEF5  LDX &F0          ..    20 DF F8
EEF7  JSR &F8DF        j     6A
EEFA  ROR A            (     28
EEFB  PLP              *     2A
EEFC  RCL A            h     68
EEFD  PLA              .     B8
EEFE  CLV              #     60
EEFF  RTS
```

Monix



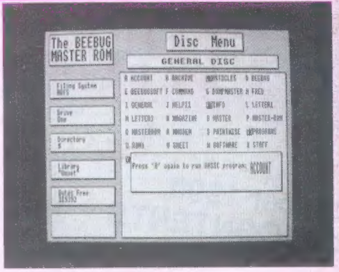
```
Date Prediction Demonstration

Enter base date
Use format (dd.mm.yyyy) 10.08.1990

Enter the interval in days 30..

The day and date 30 days ahead
will be  Sunday, 09.09.1990

Press any key to repeat
```

Pinpointing a Date

MASTER ROM

The Beebug Master ROM is a powerful 32K ROM developed to enhance the ADFS, sideways RAM, and real-time features of the Master 128 and Master Compact.

Features include: A comprehensive Disc Front End, Multi-Option Panel, Diary with Calendar and Alarm. Plus a Printer Buffer using sideways RAM, a RAM Disc, and a host of invaluable disc commands.


The BEEBUG MASTER ROM
Disc Menu
GENERAL DISC
A ACCOUNT B ARCHIVE C ARTICLES D BEEBUG
E BEEBUGSOFT F COMMAND G DUMPMASTER H FRED
I GENERAL J HELP11 K INFO L LETTER1
M LETTER2 N MAGAZINE O MASTER P MASTER-ROM
Q MASTERROM R MODEM S PAINTWISE T PROGRAMS
U ROMS V SHEET W SOFTWARE X STAFF
Filing System ADFS
Drive One
Directory $
Library "Unset"
Total Free

Press 'A' again to run BASIC program: ACCOUNT

Disc Menu

Typing *MENU takes you straight to a full feature disc menu and front panel. This gives a readout of current status and displays all the items in the current directory. Just select the file or directory of your choice to be loaded and run. The menu also allows files to be marked for subsequent copying, renaming or deleting

Disc Commands

The Master ROM boasts a wealth of new disc commands to help the user exploit the power of the ADFS with its nested directory structure. Commands include: `*BACKUP *WIPE *USED *CATALL *FIND *FORMAT *BACKUP *MERGE *DIRCOPY *FCOPY *GOTO`

Other Features

❑ **Control Panel** - displays all the Master's preset options as well as all available ROMs. The cursor keys are used to move around the panel, and to adjust the state of any items. Current status settings may be saved away to disc for subsequent reloading at any time.

❑ **Printer Buffer** - the Master ROM boasts a 16K-64K full feature printer buffer. This uses sideways RAM in selectable banks, allowing you to print long documents whilst using the computer for another task.

❑ **Diary & Alarm** - the Master ROM allows you to keep a disc-based diary and alarm with reminders for any date in the year.

❑ **RAM Disc** - a simple RAM disc of up to 64K in length. Commands are provided to save programs and memory to the RAM disc.

Members Price

£29.25

Stock Code: 0087B*

COMMAND ROM

The Command Driven Communications ROM

Unlike much communications software, this 16K ROM is command driven and has a very powerful extended command set. In addition, for ease of use, all major features are available at the touch of a function key. Because the ROM may be command driven it is exceptionally easy to link commands together in Basic to meet your own individual needs.

❑ **Text Terminal**
Use this terminal to access Telecom Gold and thousands of bulletin boards worldwide. XMODEM file transfer allows you to send files to a friend for the cost of a phone call.

❑ **Telephone Directory**
Set up the name, number and modem configuration of your favourite bulletin boards for easy recall at a later date. No need to remember telephone numbers any more, just type:

`*CALL PRESTEL`

for example, and everything will be done automatically for you - even sign-on strings and passwords are entered.

❑ **Viewdata Terminal**
A full feature Viewdata Terminal giving access to thousands of pages in Prestel and other Viewdata services.

❑ **Viewdata Editor**
A complete teletext editor, with a full range of editing commands, on-screen help, and a pixel editor.

❑ **Modem Compatibility**
The standard version of Command is suitable for the Magic Modem and similar models (Demon, Apollo etc). The Hayes version is suitable for Hayes and other intelligent modems.

Members Price

£29.25

Stock Codes:
Command 0084B*
Hayes Command 0073B*

** Please add £1.50 for P&P*

BEEBUG Ltd, 117 Hatfield Rd, St.Albans, Herts AL1 4JS. Tel. (0727)40303 Fax (0727)860263